Wei Zhang. Sentiment Analysis and Web Development of Movie Reviews Using Naïve Bayes and LSTM. A Master's Paper for the M.S. in I.S degree. April, 2020. 63 pages. Advisor: Arcot Rajasekar

Online movie reviews have become an essential reference indicator when people choose to watch movies. How to use movie reviews data to mine the audience's viewing experience and provide quantifiable reference and recommendation for other users is the problem that this paper tries to solve. This paper learns the representation of different features through the LSTM neural network model, including word-based representation, part-of-speech representation, and representation based on dependency. Then it determines and captures valid text features by fusing different feature representations to complete sentiment analysis for movie reviews. We apply this method to carry out training experiments on the IMDB dataset and compare the results with the traditional Naïve Bayes machine learning method. At the end of the research, we will use a more accurate model to develop a web to help users to estimate and choose movies that they may love. At the same time, It also could help companies and movie websites to understand the needs of the audience better and make more reasonable decisions for future movie production or distribution.

Headings:

Machine Learning

Predictive Model

Web Development

SENTIMENT ANALYSIS AND WEB DEVELOPMENT OF MOVIE REVIEWS USING NAIVE BAYES AND LSTM

by
Wei Zhang

A Master's paper submitted to the faculty
of the School of Information and Library Science
of the University of North Carolina at Chapel Hill
in partial fulfillment of the requirements
for the degree of Master of Science in
Information Science.

Chapel Hill, North Carolina

April 2020

Approved by

_____

Arcot Rajasekar

Table of Contents

1. Introduction

In recent years, with the rapid development of computers and the maturity of information technology, the Internet has gradually occupied an indispensable position in people's life, study, and work. According to statistics (Global Regional Internet Penetration Rate 2019 | Statista. (2019)), the global Internet average penetration rate up to 57% in January 2019, an increase from 35 percent in 2013. It means that there are currently 4.4 billion Internet users who use the Internet for entertainment, work, or study in the world. At the same time, it is slowly changing the various aspects of people's lives as the Internet is accessible. The most obvious change is the way of entertainment and leisure. After the network research and a large number of literature reviews, the sentiment analysis of online movie reviews has an inestimable market prospect.

The rapid development of Internet technology has made social networks gradually penetrate various fields of people's social life, and various social forums and websites have become popular. People also prefer to express their opinions on current affairs, entertainment, sports, and other events on the Internet platform. Therefore, the movie review websites represented by Rotten Tomatoes and IMDB have accumulated a large amount of user comment text data. These information data are more authentic than some companies' promotional information or advertisements, and are more representative of the user's subjective experience, and are more readily accepted by the public. By mining and analyzing the user's comment text data, we can get the user's emotional tendency, and further provide a more reliable reference for other users. At the

same time, the analysis results can help movie websites or production companies to understand better the needs of users and the prospects of certain types of movies. It also could assist them in making more reasonable decisions on the distribution of film production to ensure the maximum value of the enterprise.

2. Literature Review

As early as 1997, a personal monograph (Picard, R. (1997)) published by Professor Picard of the Media Lab of the Massachusetts Institute of Technology proposed a concept - Affecting Computing and defined as "Affecting Computing is computing that relates to, arises from, or deliberately influences emotions and or other affective phenomena.". Human emotions can embody in the form of words, sounds, and images. The primary purpose of emotional computing is to make computers have a higher level of intelligence by giving computers the ability to recognize, understand, and express human emotions. Nowadays, emotion computing is an emerging research direction, which involves the sentiment calculation of speech, sentiment calculation of images, and sentiment calculation of texts. More and more experts and scholars in the course of the speech, image, and natural language understanding have carried out extensive research on them. Specific to the field of natural language processing, text sentiment analysis methods roughly divide into two categories: based on sentiment lexicon and related information analysis and based on machine learning method analysis. In this paper, we should mainly analyze movie reviews through machine learning methods.

Theme 1: Analysis Based on Traditional Machine Learning Methods

The method of emotion classification, initially proposed by Pang, B., Lee, L., & Vaithyanathan, S. (2002), is mainly applied to online reviews. They extracted features according to different methods such as Unigram, Bigram, POS, Position, etc. Then they

compared and analyzed the experimental results of Support Vector Machine (SVM), Expectation-Maximization Algorithm (EM), and Naïve Bayes (NB) three machine learning methods. And finally, they found the extraction method of SVM combined with Unigram features is the best on the IMDB movie review dataset, and the correct rate reached 82.7%. In their experiment, SVM was an appropriate tool because it was resistant to blog noise. It can handle large feature sets consisting of a bag of unigram and bigram words feature sets, and it was traditionally good at similar tasks like topic-based classification.

Whitelaw, C., Garg, N., & Argamon, S. (2005) extracted adjective phrases in movie reviews and combined standard lexicon feature representations. They used vector space models to represent text and support vector machine methods for classification and finally distinguished positive and negative comments Information. The accuracy rate of their approach increased to 90.2%. Boiy, E., & Moens, M. (2008) experimented with Dutch, French, and English in a machine learning method. The results showed that the accuracy rates of the emotional classification of three languages were respectively 70% and 68%, 83%. So we can know that machine learning methods have certain advantages in foreign language sentiment analysis.

Turney, P. (2001) proposed a method based on pointwise mutual information value to analyze the emotional polarity of a specific phrase and further estimated the psychological orientation of the whole comment. The process firstly divided the text into words, labeled their parts of speech, and extracted the adjective or adverb phrase. Then they used the mutual information between the two words "excellent" & "poor" and the

unknown word in the search page to calculate the emotional polarity of the unknown word. Finally, they could use the result to calculate the emotional polarity of the entire text.

All of the above are based on traditional machine learning methods. Through a comprehensive study and reference to the conventional machine learning methods of literature, we will use the Naïve Bayes machine learning method to complete the movie review sentiment analysis in this paper.

Theme 2: Analysis Based on Deep Learning Methods

In recent years, with the rapid development of deep learning and neural networks, Teng, Z., Vo, D., & Zhang, Y. (2016) established a neural network model based on contextual sentiment lexicon to conduct the emotion classification. The results showed that the model could further improve the accuracy rate of the classification. Qian Q., Huang, M., Lei, J., & Zhu, X. (2016) proposed a simple model of sentence-level annotation training, which could capture the language characters in the emotional expression of the emotional vocabulary, the negative vocabulary and the intensity vocabulary. Finally, they have achieved excellent results in emotional classification.

Recently, with the rapid development of new media such as Twitter and Facebook, more and more people started sentiment analysis for short texts like Twitter comments. Cliche, M. (2017) produced a state-of-the-art Twitter sentiment classifier using Convolutional Neural Networks (CNNs) and Long Short Term Memory (LSTM) networks. His system

leveraged a large amount of unlabeled data to pre-train word embeddings. And then, he used a subset of the unlabeled data to fine-tune the embeddings using distant supervision.

Pong-inwong, C., & Songpan, W. (2018) proposed a new sentiment analysis method - Sentiment Phrase Pattern Matching (SPPM). Compared with other algorithms, the accuracy, recall rate, and F1 value of SPPM were greatly improved, and this method also could develop the teaching strategy according to the students' opinions. Shen, C., & J, J. (2018) proposed a text sentiment classification algorithm based on extended features and dynamic merging of two-channel convolutional neural networks. This algorithm had a better classification effect than the traditional single-channel convolutional neural networks.

Through a comprehensive study and reference to the deep learning methods of literature, we will use Long Short Term Memory (LSTM) networks to complete the movie review sentiment analysis in this paper.

Theme 3: Relevant Applications of Text Sentiment Analysis

Developed by Dave, K., Lawrence, S., & Pennock, D. (2003), Revier Seer was the first sentiment analysis system that automatically analyzed the ambiguity of given product-related reviews. The Pulse system developed by Gamon, M., Aue, A., Corston-Oliver, S., & Ringger, E. (2005) could automatically mine the emotional sentiment and strength of uploaded car evaluation texts. The Opinion Finder system developed by T. Wilson, P.

Hoffmann. (2005) automatically recognized subjective statements and subjective-related information in comments.

Jeonghee Yi, & Niblack, W. (2005) developed the opinion mining tool in the Web Fountain system. The Web Fountain system is a research and development platform based on multi-type data and open domain opinion mining. The Opinion Observer system developed by Liu, B., Hu, M., & Cheng, J. (2005) could count emotional tendencies of characteristics of products from a large number of evaluations published by product users in online forums, and then visually compare the quality of the main features of the product. Bosco, C., Patti, V., & Bolioli, A. (2013) researched the sentiment software developed by Corpora, UK, which analyzed the articles published in newspapers to determine the attitude of authors for the policy of a specific political party.

Through a comprehensive study and reference to development methods of these software applications, we will also try to develop the model of movie review sentiment analysis into a corresponding web, which could be convenient and helpful for users to estimate and choose movies that they may love.

3. Methodology

3.1 Data Collection and Preparation

The dataset used we is the IMDB movie reviews dataset provided by the Bag of Words Meets Bags of Popcorn program of the Kaggle website. The data set consists of 50,000 IMDB movie reviews, specially selected for sentiment analysis. The sentiment of reviews is binary, meaning the IMDB rating < 5 results in a sentiment score of 0, and rating >=7 has a sentiment score of 1. No individual movie has more than 30 reviews. The 25,000 labeled review training set does not include any of the same films as the 25,000 review test set. The training set and the test set share two possible features for analysis, and the only difference between these two sets is that there is one more label in the training set indicating the sentiment score of the review but not in the test set.

The specific descriptions of data set files are listed in detail below:

- **labeledTrainData** - The labeled training set. The file is tab-delimited and has a header row followed by 25,000 rows containing an id, sentiment, and text for each review.
- **testData** - The test set. The tab-delimited file has a header row followed by 25,000 rows containing an identifier and content for each review. Your task is to predict the sentiment for each one.

The specific features' names and their corresponding descriptions are listed in detail in the tables below:

Field	Field Description
id	Unique ID of each review
sentiment	The sentiment score of the review; 1 for positive reviews and 0 for negative reviews
review	Text of the review

The labeledTrainData and the testData datasets were already quite well-organized, but we still need to preprocess datasets, including removing labels, symbols, and stop words. We will introduce this part in detail in the Model section. We need to build and train our models with the labeledTrainData dataset firstly, and then test models with the testData dataset and calculate a few evaluation indicators of models.

3.2 Traditional Machine Learning Method - Naïve Bayes

3.2.1 Bayes' Theorem

Bayesian classification is a general term for a class of classification algorithms. These algorithms are based on Bayes' theorem and are collectively referred to as Bayesian classification. Naïve Bayes classification used in this paper is also one of them.

First, for the classification problem, from a mathematical perspective, it can be defined as follows:

Given the set: $C = \{y_1, y_2, ..., y_n\}$ and $I = \{x_1, x_2, ..., x_m\}$, determine the mapping rules $y = f(x)$, so that any $x_i \in I$ has only one $y_j \in C$ to make $y_j = f(x_i)$ true.

Where C is called the categories set, and each element is a category. Where I is called the items set and each element is an item to be classified, and f is called a classifier. In the classification algorithm, our task is to construct a classifier f, that is, to build a model. However, we often use empirical methods to construct mapping rules in classification problems. Because we usually lack enough information to develop 100% correct mapping rules, and we could only achieve correct classification in a certain probability by learning from empirical data. Therefore, the trained classifier may not be able to map all Element accurately, and the quality of the classifier is related to many factors such as the construction method of the classifier, the characteristics of the data to be classified, and the number of training samples.

Then we need to understand the conditional probability, that is, given a conditional probability, how to obtain the probability after the exchange of two events, that is, how to find P (B | A) when P (A | B) is known. P (A | B) is a conditional probability: the likelihood of event A occurring given that B is true. The conditional probability is stated mathematically as the following equation (Equation 3.1), and we also can derive the Bayes' theorem (Equation 3.2) by the conditional probability formula.

$$P(A \mid B) = \frac{P(AB)}{P(B)} \quad (3.1) \qquad P(B \mid A) = \frac{P(A \mid B)P(B)}{P(A)} \quad (3.2)$$

3.2.2 Naïve Bayes Classification

Naïve Bayes is a straightforward and practical classification algorithm in Bayesian classification. Its ideological basis can be summarized as follows: For a given item to be classified, calculate the probability of the item appearing in each category, whichever is

the largest, consider that the item to be classified belongs to which category. The formal definition of the Naïve Bayes classification is as follows:

1. Suppose $x = \{a_1, a_2, ..., a_m\}$ is an item to be classified, and each a is a feature attribute of x.

2. There is a category set $C = \{y_1, y_2, ..., y_n\}$.

3. Calculate $P(y_1 \mid x), P(y_2 \mid x), ..., P(y_n \mid x)$.

4. If $P(y_k \mid x) = \max\{P(y_1 \mid x), P(y_2 \mid x), ..., P(y_n \mid x)\}$, then $x \in y_k$.

So we can know that the key to the problem is how to calculate the conditional probabilities in the third step. We can try to do this:

1. Find a set of items to be classified with known classification, and this set is called the training sample set.

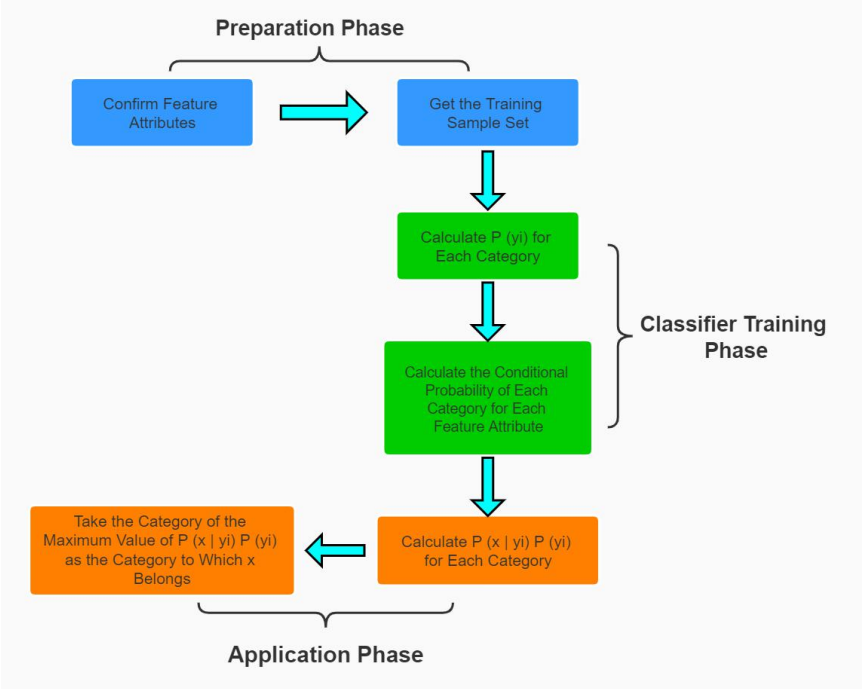
2. Calculate the conditional probability of each feature attribute of the items in each category that is:

$$P(a_1 \mid y_1), P(a_2 \mid y_1), ..., P(a_m \mid y_1); P(a_1 \mid y_2), P(a_2 \mid y_2), ..., P(a_m \mid y_2); ...; P(a_1 \mid y_n), P(a_2 \mid y_n), ..., P(a_m \mid y_n)$$

3. If each feature attribute is conditionally independent, we can derive as follows, according to Bayes' theorem: $P(y_i \mid x) = \frac{P(x \mid y_i)P(y_i)}{P(x)}$.

Because the denominator is constant for all categories, we only need to maximize the numerator. At the same time, each feature attribute is conditionally independent, there are: $P(x \mid y_i)P(y_i) = P(a_1 \mid y_i)P(a_2 \mid y_i)...P(a_m \mid y_i)P(y_i) = P(y_i)\prod_{j=1}^{m} P(a_j \mid y_i)$.

Based on the above analysis, the process of Naïve Bayes classification can be represented by the following figure:



So we can divide the entire Naïve Bayes classification into three phases:

- **Preparation Phase.** The task of this phase is to make the necessary preparations for the Naïve Bayes classification. The main task is to determine the feature attributes according to the specific situation, to rightly divide each feature attribute, and then to classify some items to form a training sample set. The input at this phase is all the data to be classified, and the output is the feature attributes and training sample sets. The quality at this phase will have a significant impact on the entire process because the quality of the classifier is primarily determined by the feature attributes, the feature attribute division, and the quality of the training sample set.

- **Classifier Training Phase.** The task of this phase is to generate a classifier. The main task is to calculate the frequency of occurrence of each category in the training sample set and the conditional probability of each feature attribute for each group. Its input is feature attributes and the training sample set, and its output is a classifier.
- **Application Phase.** The task of this phase is to use the classifier to classify the items to be classified. The input is the classifier and the item to be classified, and the output is the mapping relationship between the item to be classified and the category.

3.2.3 Naïve Bayes Classification Application

In this paper, we need to build a Naïve Bayes classification classifier to classify text for further sentiment analysis. How will we apply the Naïve Bayes classification during the movie reviews analysis? Here is an example to introduce.

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable with no fun

From the above introduction of Naïve Bayes classification, we can know that we only need to maximize the molecules of Naïve Bayes' theorem to get the emotional bias of the

test data, which is:

$$P(x \mid y_i)P(y_i) = P(\mathrm{a}_1 \mid y_i)P(\mathrm{a}_2 \mid y_i)...P(\mathrm{a}_m \mid y_i)P(y_i) = P(y_i)\prod_{j=1}^{m} P(a_j \mid y_i)\,.$$

$$P(positive) = \frac{2}{5} \qquad P(\mathrm{negative}) = \frac{3}{5}$$

We can choose the probability of the word appearing in all documents as the feature attribute. However, sometimes some words may never appear in all documents. Therefore, we usually need to make some small modifications such as Laplace smoothing, which adds 1 to all counts.

$$P(\text{"}predictable\text{"} \mid -) = \frac{1+1}{14+20} \qquad P(\text{"}predictable\text{"} \mid +) = \frac{0+1}{9+20}$$

$$P(\text{"}no\text{"} \mid -) = \frac{1+1}{14+20} \qquad P(\text{"}no\text{"} \mid +) = \frac{0+1}{9+20}$$

$$P(\text{"fun"} \mid -) = \frac{0+1}{14+20} \qquad P(\text{"}fun\text{"} \mid +) = \frac{1+1}{9+20}$$

$$P(\text{"}with\text{"} \mid -) = \frac{0+1}{14+20} \qquad P(\text{"}with\text{"} \mid +) = \frac{0+1}{9+20}$$

For the test set S = "predictable with no fun", the probabilities in positive and negative are calculated as follows:

$$P(-)P(S \mid -) = \frac{3}{5} \times \frac{2 \times 2 \times 1 \times 1}{34^4} = 1.79 \times 10^{-6} \quad P(+)P(S \mid +) = \frac{2}{5} \times \frac{1 \times 1 \times 2 \times 1}{29^4} = 1.1 \times 10^{-6}$$

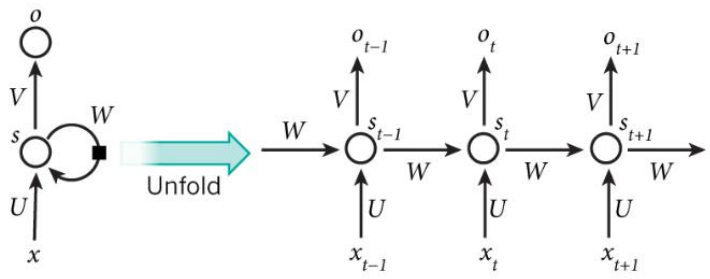
Because the probability of negative is higher than the probability of positive, the sentiment of the test set is predicted to be negative through the Naïve Bayes classification model.

3.3 Deep Learning Method - LSTM Neural Network

3.3.1 Recurrent Neural Network - RNN

Recurrent Neural Network (RNN) is a neural network for processing sequence data. Compared with the general neural network, it can process the data of sequence change. For example, the meaning of a word will be different because of the different content above, and RNN can solve this type of problem well.

In the Artificial Neural Network, the information is from the input layer to the hidden layer to the output layer output. The layers are fully connected, and the nodes between each layer are disconnected. But this ordinary neural network is powerless for many problems. For example, when you want to predict the next word of a sentence, you generally need to use the previous word, because the preceding and following words in a sentence are not independent. In RNN, the current output of a sequence is related to the previous output. The specific performance is that the network will remember the previous information and apply it to the calculation of the current output; that is, the nodes between the hidden layers are connected. And then the input of the hidden layer includes not only the output of the input layer but also the output of the hidden layer at the last time. In theory, RNN can process the sequence data of any length. However, it is often assumed that the current state is only related to the previous states in practice so that it could reduce complexity. The following figure is a typical RNN:



3.3.2 Long Short Term Memory Network - LSTM

The Long Short Term Memory network is a special RNN that can learn long-term dependent information. LSTM was proposed by Hochreiter and Schmidhuber (Hochreiter, S., & Schmidhuber, J. (1997)), and was recently improved and promoted by Alex Graves. LSTM has achieved considerable success on a large variety of problems and has been widely used.

All recurrent neural networks have the form of a chain of repeating modules of a neural network. In a standard RNN, this repeated module has only a straightforward structure, such as a tanh layer (Figure 3.1). LSTM also has such a structure, but its repeated modules have a different structure. Instead of having a single neural network layer, there are four (Figure 3.2), interacting in a very special way.

The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to let information through optionally. They contain a sigmoid neural network layer and a pointwise multiplication operation (Figure 3.3). The Sigmoid layer outputs a value between 0 and 1, describing how much of each component should be let through. 0 means “let nothing through”, 1 means “let everything through”. LSTM has three gates to protect and control the cell state.

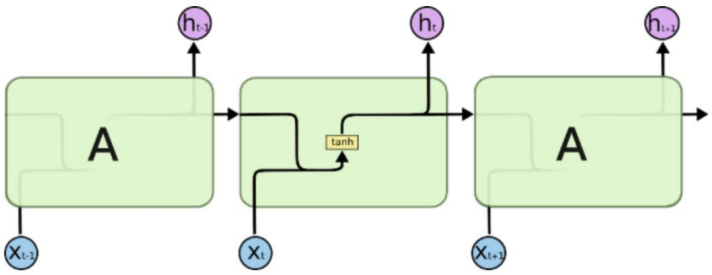


Figure 3.1 The repeating module in a standard RNN contains a single layer.

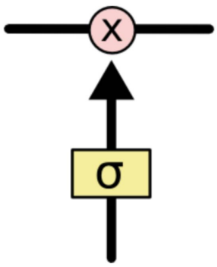


Figure 3.3 A Gate

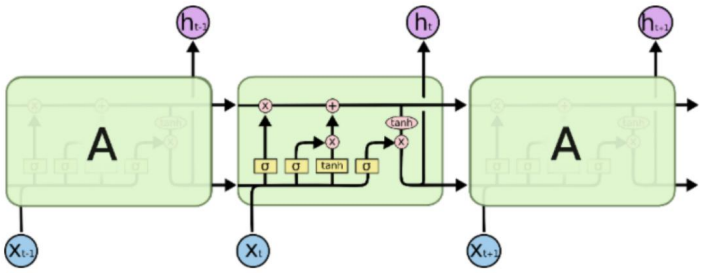
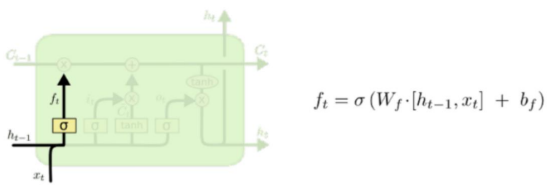


Figure 3.2 The repeating module in an LSTM contains four interacting layers.

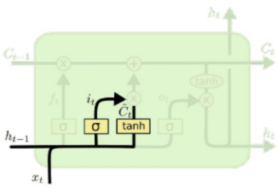
3.3.3 The Core Idea Behind LSTM

The first step in LSTM is to decide what information we need to discard. This decision is made through a "forget gate layer". The gate will input h_{t-1} and x_t and output a value between 0 and 1 for each number in the cell state C_{t-1} (Figure 3.4). 1 means "completely keep this", 0 means "completely get rid of this".



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

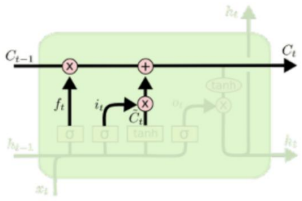
Figure 3.4



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \sigma\left(W_C \cdot [h_{t-1}, x_t] \; + \; b_C\right)$$

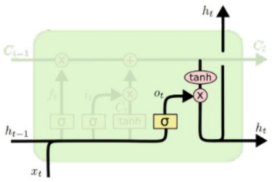
Figure 3.5

The second step is to determine the new information that needs to be stored in the cell state. There are two parts here. Firstly, a sigmoid layer called the “input gate layer” decides the value we need to update. Then, a tanh layer creates a new candidate value vector $\widetilde{C}_t$, which will be added to the cell state (Figure 3.5). Secondly, we combine two parts to update the cell state C_{t-1} to C_t. We multiply the old state C_{t-1} by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \widetilde{C}_t$ to get new candidate values, scaled by how much we decided to update each state value (Figure 3.6).



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 3.6

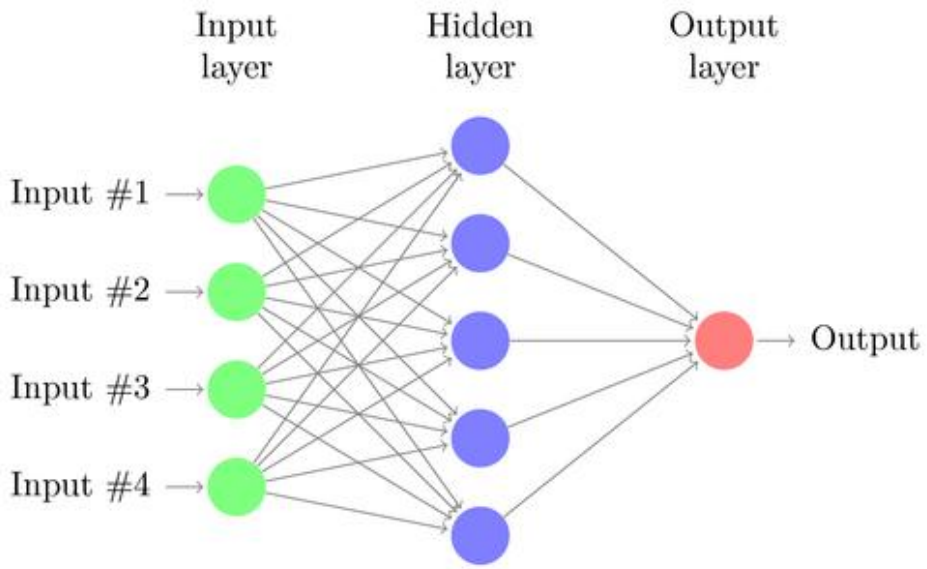


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 3.7

In the final step, we need to determine the output information. This output will be based on our cell state, but it is also a filtered version. Firstly, we run a sigmoid layer to determine the output part of the cell state. Then, we process the cell state through tanh (to push the values to be between −1 and 1) and multiply it with the output of the sigmoid gate, so that we will only output the part we determined to output (Figure 3.7).

The complete flow chart of the LSTM classifier we constructed is shown below.



4. Model

4.1 Introduction

The section introduces two models that will be experimented in the study, Logistic Regression, and Gradient Boosting. Both of the models are presented in detail with deep consideration in terms of model establishment and system optimization. Their advantages, disadvantages, and comparison are also explained at the end of this section.

4.2 Naïve Bayes Modeling

4.2.1 Data Preprocessing

Firstly, we first need to find out the data packages and libraries needed during the experiment, download, and install them before we start to build the model. Our experiments involve deep learning algorithms, so some existing learning libraries and algorithms can help us to improve the coding efficiency and accuracy of experiments significantly. In this experiment, the data packages and libraries we need are Pandas, NumPy, sklearn, nltk and matplotlib.

```
#import pandas, numpy, sklearn, Nltk and matplotlib package
import pandas as pd
import numpy as np
from sklearn.naive_bayes import MultinomialNB
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer,TfidfTransformer,TfidfVectorizer
import nltk
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection   import train_test_split
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer as TFIDF
from sklearn.metrics import accuracy_score,confusion_matrix,precision_recall_curve,average_precision_score
from sklearn import metrics
from pylab import *
import numpy as np
from sklearn.metrics import roc_curve, auc
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.externals import joblib
import re
from sklearn.metrics import classification_report
nltk.download('stopwords')
```

Secondly, we need to perform preliminary text processing on the data set. We need to extract the reviews part and the scores part in the data set separately. At the same time, for the reviews part, we should remove some unnecessary symbols and labels, keep only the English words, and then convert them to lowercase. We could save the processed reviews in an array format for further processing.

```
def review_to_wordlist(review):
    #Only save English words
    review_text = re.sub("[^a-zA-Z]"," ", review)
    #to lower case
    words = review_text.lower()
    return(words)

# Separate the reviews and sentiments
# 1. extract the sentiments
y_train_1 = train['sentiment']
# 2. extract the reviews
train_data = []
for review in train['review']:
    train_data.append(review_to_wordlist(review))
# 3. transfer to numpy array
train_data = np.array(train_data)
```

After the processing is completed, we get the review array and sentiments series.

```
train_data
```

```
array(['with all this stuff going down at the moment with mj i ve started listening to his music  watching the odd documentary here and ther
e  watched the wiz and watched moonwalker again  maybe i just want to get a certain insight into this guy who i thought was really cool in t
he eighties just to maybe make up my mind whether he is guilty or innocent  moonwalker is part biography  part feature film which i remember
going to see at the cinema when it was originally released  some of it has subtle messages about mj s feeling towards the press and also the
obvious message of drugs are bad m kay  br  br  visually impressive but of course this is all about michael jackson so unless you remotel
y like mj in anyway then you are going to hate this and find it boring  some may call mj an egotist for consenting to the making of this mov
ie but mj and most of his fans would say that he made it for the fans which if true is really nice of him  br  br  the actual feature fil
m bit when it finally starts is only on for  minutes or so excluding the smooth criminal sequence and joe pesci is convincing as a psychop
athic all powerful drug lord  why he wants mj dead so bad is beyond me  because mj overheard his plans  nah  joe pesci s character ranted th
at he wanted people to know it is he who is supplying drugs etc so i dunno  maybe he just hates mj s music  br  br  lots of cool things i
n this like mj turning into a car and a robot and the whole speed demon sequence  also  the director must have had the patience of a saint w
hen it came to filming the kiddy bad sequence as usually directors hate working with one kid let alone a whole bunch of them performing a co
mplex dance scene  br  br  bottom line  this movie is for people who like mj on one level or another  which i think is most people  if n
ot  then stay away  it does try and give off a wholesome message and ironically mj s bestest buddy in this movie is a girl  michael jackson
is truly one of the most talented people ever to grace this planet but is he guilty  well  with all the attention i ve gave this subject
hmmm well i don t know because people can be different behind closed doors  i know this for a fact  he is either an extremely nice but stupi
d guy or one of the most sickest liars  i hope he is not the latter ',
       ' the classic war of the worlds  by timothy hines is a very entertaining film that obviously goes to great effort and lengths to fai
thfully recreate h  g  wells  classic book  mr  hines succeeds in doing so  i  and those who watched his film with me  appreciated the fact
that it was not the standard  predictable hollywood fare that comes out every year  e g  the spielberg version with tom cruise that had only
the slightest resemblance to the book  obviously  everyone looks for different things in a movie  those who envision themselves as amateur
critics  look only to criticize everything they can  others rate a movie on more important bases like being entertained  which is why most
people never agree with the  critics  we enjoyed the effort mr  hines put into being faithful to h g  wells  classic novel  and we found
it to be very entertaining  this made it easy to overlook what the  critics  perceive to be its shortcomings ',
```

```
y_train_1
```

0	1
1	1
2	0
3	0
4	1
5	1
6	0
7	0
8	0
9	1
10	0
11	1
12	1
13	0
14	0
15	0
16	0
17	0
18	1
19	1
20	1
21	1
22	1

4.2.2 TF-IDF

4.2.2.1 TF-IDF Method

TF-IDF (term frequency–inverse document frequency) is a commonly used weighting method for information retrieval and data mining. It is often used to mine keywords in articles, and the algorithm is straightforward and efficient. TF-IDF has two meanings, one is "Term Frequency" (abbreviated as TF), and the other is "Inverse Document Frequency" (abbreviated as IDF).

- **TF**: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization (Tf-idf :: A Single-Page Tutorial -

Information Retrieval and Text Mining. (2020)):

$$TF(t) = \frac{Number\ of\ times\ term\ t\ appears\ in\ a\ document}{Total\ number\ of\ terms\ in\ the\ documents}$$

- **IDF**: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However, it is known that specific terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scaling up the rare ones, by computing the following (Tf-idf :: A Single-Page Tutorial - Information Retrieval and Text Mining. (2020)):

$$IDF(t) = \log \frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ t\ in\ it}$$

The more common a word, the larger the denominator of the formula, and the smaller the inverse document frequency, the closer to zero. The denominator is increased by 1 to avoid the denominator being 0 (that is, all documents don't contain this word).
When we multiply two frequency TF and IDF, we can get the TF-IDF value of one word. The larger the TF-IDF of a word in the article, the higher the importance of the word in this article in general. So we can rank these words in descending order by calculating the TF-IDF of each word in the article. The first few words in the order are the keywords of the article.

4.2.2.2 Stop Words

We also need to remove stop words. A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query (Removing

stop words with NLTK in Python - GeeksforGeeks. (2020)). We would not want these words to take up space in our database, or to take up the valuable processing time. NLTK (Natural Language Toolkit) in python has a list of stop words stored in 16 different languages. We can find them in the nltk_data directory and remove the English stop words in our code.

4.2.2.3 TF-IDF Application - TfidfVectorizer

In this experiment, we use the TF-IDF method to extract text features and use the TfidfVectorizer tool in python to perform words segmentation on all samples. We also set the N-gram to obtain the features, and then use these words as dimensional features for each sample to perform vectorization, and finally train them in the model.

```
tfidf = TfidfVectorizer(ngram_range=(1, 2), min_df=0.01, use_idf=1,smooth_idf=1, stop_words = 'english') # remove English stop words
data_train_count = tfidf.fit_transform(train_data)
```

```
words = tfidf.get_feature_names()
print(words[1398])
print(words[609])
print(words[950])
```

```
stuff
going
moment
```

```
print(data_train_count)
```

```
  (0, 1398)     0.07359370363384517
  (0, 609)      0.15836051874700902
  (0, 950)      0.07480337141431258
  (0, 1546)     0.09991275705206315
  (0, 1370)     0.07651748600849363
  (0, 982)      0.11911861384547107
  (0, 1586)     0.05061903444777294
  (0, 1022)     0.08615487797747001
  (0, 381)      0.08259726086604724
  (0, 1584)     0.12453812750958633
  (0, 912)      0.1867271152132105
  (0, 769)      0.09834918670435419
  (0, 1570)     0.054305313363520784
  (0, 207)      0.08049564793422298
  (0, 634)      0.12033008209916535
  (0, 1472)     0.05540369074399123
  (0, 1182)     0.0769957505715714
  (0, 288)      0.15500970061833427
  (0, 889)      0.04233447422070196
  (0, 938)      0.06382262731867913
  (0, 732)      0.09125367362228724
  (0, 506)      0.16099129586844596
  (0, 529)      0.05576281423791415
  (0, 1203)     0.06773855562986225
  (0, 234)      0.07108707381100518
```

We can use the TfidfVectorizer tool to remove stop words from the text, and calculate the TF-IDF value of the words in the document to get the data_train_count matrix. From the above code, we can see that in the first document, the TF-IDF value of the “stuff” word is 0.0736, the TF-IDF value of the “going” word is 0.1584, and the TF-IDF value of the “moment” word is 0.0748.

4.2.3 MultinomialNB () Function Modeling

The train_test_split is a commonly used function in the cross-validation. Its function is to randomly compose training dataset train_data and test dataset test_data from the processed samples. In this experiment, we randomly select train_data and test_data according to the ratio of 0.2, that is, 20,000 data in train_data and 5000 data in test_data.

In the scikit-learn machine learning library, there are three Naïve Bayes classification algorithms. They are GaussianNB, MultinomialNB, and BernoulliNB. Bernoulli Naïve Bayes assumes that all our features are binary such that they take only two values. Means 0s can represent "word does not occur in the document" and 1s as "word occurs in the document". Multinomial Naïve Bayes is used when we have discrete data (e.g. movie ratings ranging 1 and 5 as each rating will have a specific frequency to represent). In the text learning, we have the count of each word to predict the class or label. Because of the assumption of the normal distribution, Gaussian Naïve Bayes is used in cases when all our features are continuous (1.9. Naïve Bayes — scikit-learn 0.22.2 documentation. (2020)). These three methods are applicable to different classification scenarios. In this experiment, we choose to use MultinomialNB. Firstly, we use the fit method to fit the classified train_data dataset and obtain the prediction result Pred of the model for X_test dataset. We can find that the difference between these two datasets is small by calculating and comparing the scores of the

train_data dataset and the test_data dataset. So, we think that the model and the classifier we established are not overfitting.

```
X_train, X_test, y_train, y_test = train_test_split(data_train_count, y_train_1, test_size = 0.2, random_state = 0)
```

```
#multinomialNB model
clf = MultinomialNB()
clf.fit(X_train, y_train)
pred = clf.predict(X_test)
```

```
#score of test dataset
clf.score(X_test, y_test)
```

```
0.8498
```

```
#score of train dataset
clf.score(X_train, y_train)
```

```
0.85035
```

Through the predict_proba method, the model will predict the results of the test_data dataset and calculate the prediction probability of the corresponding results. That is, in the first review, the model estimates that its probability of negative is 0.49, and the probability of positive is 0.51. At the same time, we can conclude that the accuracy of the model is 0.8498 through the accuracy_score method.

```
#probability estimates for the test
clf.predict_proba(X_test)
```

```
array([[0.48816433, 0.51183567],
       [0.45184684, 0.54815316],
       [0.27062664, 0.72937336],
       ...,
       [0.40218761, 0.59781239],
       [0.84719866, 0.15280134],
       [0.2512578 , 0.7487422 ]])
```

```
# accuracy of model
accuracy_score(y_test, pred)
```

```
0.8498
```

4.2.4 Cross-Validation

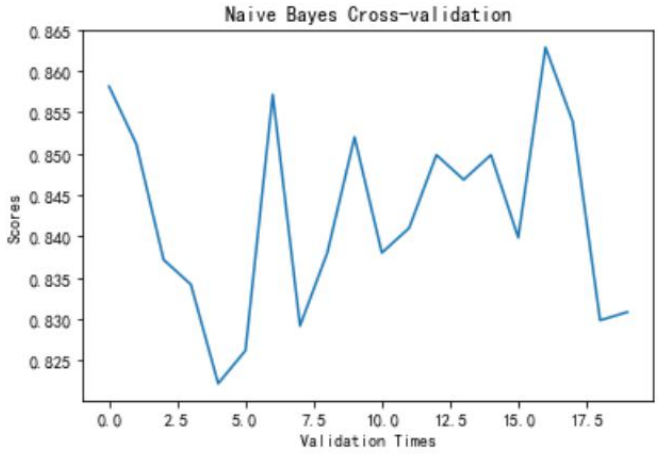
Cross-validation is any of various similar model validation techniques for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (called the validation dataset or testing set). The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting or selection bias and to give an insight on how the model will generalize to an independent dataset (Cross-validation (statistics). (2020)).

In the experiment, we perform 20 cross-validations and plot the scores obtained from each verification as a graph. From the chart, we can see that the accuracy of the model is distributed between 0.83 and 0.855 after removing individual extreme points, which is relatively stable. We could conclude that the accuracy of the model is 0.8424 by calculating the average of 20 verifications. The difference between the accuracy rate of 0.8498 we obtained in the previous section and 0.8424 is very small. So, we can think that the accuracy of the model is 0.85.

```
times = 20
scores = cross_val_score(clf, X_train, y_train, cv=times)
```

```
mpl.rcParams['font.sans-serif'] = ['SimHei']
plt.title('Naive Bayes Cross-validation')
plt.xlabel("Validation Times")
plt.ylabel("Scores")
plt.plot(range(times), scores)
```

```
[<matplotlib.lines.Line2D at 0x2049f84e208>]
```




```
avgscore = np.mean(scores)
print(avgscore)
```

```
0.8424024239524239
```

4.2.5 Various Indicators

4.2.5.1 Confusion Matrix

A confusion matrix is a situation analysis table that summarizes the prediction results of a classification model in machine learning. It summarizes the records in a dataset in two dimensions ("actual" and "predicted") in a matrix. Each row of the matrix represents the instances in a predicted class, while each column represents the instances in an actual class (or vice versa). It makes it easy to see if the system is confusing two classes (i.e. commonly mislabeling one as another). From the graph and table below, we compare the results predicted by the model pred with the actual test_data dataset results, and we can

see that the model predicted that there are 2124 negatives and 424 positives of the actual negative results. There are 327 negatives and 2125 positives of the actual positive results.

```
#confusion matrix
confusion_matrix(y_test, pred, labels=None, sample_weight=None)
```

```
array([[2124,  424],
       [ 327, 2125]], dtype=int64)
```

Actual Predict	Negative	Positive
Negative	2124	424
Positive	327	2125

4.2.5.2 ROC Curve

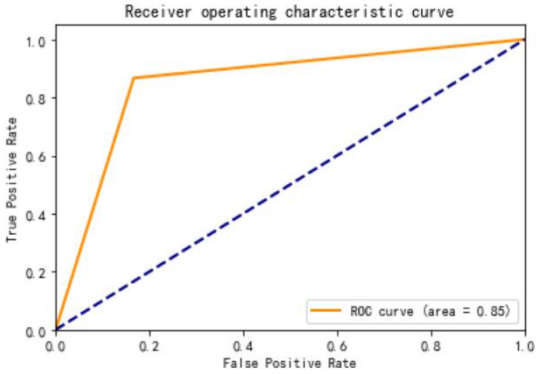
A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. It can also be thought of as a plot of the power as a function of the Type I Error of the decision rule (Receiver operating characteristic. (2020)).

The full name of the ROC curve is the "Receiver Operating Characteristic" curve, which is derived from the radar signal analysis technology used for enemy aircraft detection in World War II. It is a comprehensive indicator reflecting sensitivity and specificity. It calculates a series of sensitivity and specificity by setting continuous variables to a number of different critical values. It then draws a curve with sensitivity as the vertical axis and (1-specificity) as the horizontal axis. The larger the area under the curve, the higher the accuracy of the discrimination. On the ROC curve, the point closest to the upper left of the graph is the critical value with higher sensitivity and specificity.

From the ROC curve, we can see that the area under the curve is very high. We can also think that the accuracy of the model is excellent by calculating the AUC area as 0.8501.

```
fpr, tpr, thresholds = metrics.roc_curve(y_test, pred, pos_label=1)
```

```
#ROC curve
import matplotlib.pyplot as plt
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',lw=lw, label='ROC curve (area = %0.2f)' % accuracy_score(y_test, pred))
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic curve')
plt.legend(loc="lower right")
plt.show()
```




```
# auc area
metrics.auc(fpr, tpr)
```

```
0.8501172272146404
```

4.2.5.3 Precision, Recall, and F-matures indicators

The following conclusions can be drawn from the calculation. For the negative class, the precision is 0.87, the recall is 0.83, and the F1 threshold is 0.85. For the positive class, the precision is 0.83, the recall is 0.87, and the F1 threshold is 0.85. A comprehensive comparison of the two classes shows that the precision, recall, and F1 thresholds of this model are all 0.85.

```
print(classification_report(y_test, pred))
```

	precision	recall	f1-score	support
0	0.87	0.83	0.85	2548
1	0.83	0.87	0.85	2452
micro avg	0.85	0.85	0.85	5000
macro avg	0.85	0.85	0.85	5000
weighted avg	0.85	0.85	0.85	5000

In summary, the various indicators and verification methods show that the accuracy of the Naïve Bayes classification model we build is very high and stable. We will compare and analyze the Naïve Bayes classification model and the LSTM model in the subsequent sections. The complete Naïve Bayes model code is located in Appendix A.

4.3 LSTM Modeling

4.3.1 Data Preprocessing

Firstly, we first need to find out the data packages and libraries needed during the experiment, download, and install them before we start to build the model. Our experiments involve deep learning algorithms, so some existing learning libraries and algorithms can help us to improve the coding efficiency and accuracy of experiments significantly. In this experiment, the data packages and libraries we need are pandas, NumPy, sklearn, nltk, matplotlib, Keras and TensorFlow.

Secondly, we need to perform preliminary text processing on the data set. We define a function review_to_words to perform text processing on the reviews part of the data set, including removing Html labels and special characters, converting to lowercase, removing stop words, lemmatize words, and joining them, and finally obtaining the text of the cleaned reviews.

```
def review_to_words( raw_review ):
    #remove html label
    review_text = BeautifulSoup(raw_review).get_text()
    #remove special characters
    letters_only = re.sub("[^a-zA-Z]", " ", review_text)
    #to lower case
    words = letters_only.lower().split()
    #remove stop words
    stops = set(stopwords.words("english"))
    meaningful_words = [w for w in words if not w in stops]
    # lemmatize list of words and join them
    for word in meaningful_words:
        word = wordnet_lemmatizer.lemmatize(word,'v')
    return( " ".join( meaningful_words))
```

```
#remove all labels of the dataset, lemmatize list of words and join them
train['Phrase_clean_review'] = train.apply(lambda x: review_to_words(x['review']), axis=1)
train.head()
```

	id	sentiment	review	Phrase_clean_review
0	5814_8	1	With all this stuff going down at the moment w...	stuff going moment mj started listening music ...
1	2381_9	1	\The Classic War of the Worlds\" by Timothy Hi...	classic war worlds timothy hines entertaining ...
2	7759_3	0	The film starts with a manager (Nicholas Bell)...	film starts manager nicholas bell giving welco...
3	3630_4	0	It must be assumed that those who praised this...	must assumed praised film greatest filmed oper...
4	9495_8	1	Superbly trashy and wondrously unpretentious 8...	superbly trashy wondrously unpretentious explo...

Finally, we need to separately extract the cleaned reviews part in the dataset as train_text and the score part as the target, and converts a class vector (integers) to the binary class matrix through the to_categorical function in Keras. With the help of the nltk.tokenize.word_tokenize () method, we are able to get a tokenized copy of reviews text and generate a FreqDist object. The keys in FreqDist are all unique words in the reviews text, and the value is the total number of occurrences of the corresponding word. For example, we can see that the "delegate" appears once, and the "dashes" appeared 9 times. We can get the number of unique words by calculating the length of FreqDist object is 74064. At the same time, we can get the specific word count of each review in the dataset, so as to find the review with the most word count has 1416 words.

```
train_text=train.Phrase_clean_review.values
target=train.sentiment.values
y=to_categorical(target)
all_words=' '.join(train_text)
all_words=word_tokenize(all_words)
dist=FreqDist(all_words)
for key in dist:
    print(key, dist[key])
num_unique_word=len(dist)
```

```
delegate 1
unlockables 1
schnooks 1
macdowel 1
macrae 3
seasoning 1
dashes 9
coffins 6
seens 2
myrnah 1
daisy 56
witchie 1
thepeople 1
pedicure 1
pothus 1
crappily 1
pavignano 1
invasive 2
intuition 5
hydrogen 7
```

```
num_unique_word
```

```
74064
```

```
r_len=[]
for text in train_text:
    word=word_tokenize(text)
    l=len(word)
    r_len.append(l)
for i in r_len:
    print(i)
```

```
219
84
240
189
210
43
55
69
93
23
35
71
200
71
58
103
213
224
138
```

```
MAX_REVIEW_LEN=np.max(r_len)
print(MAX_REVIEW_LEN)
max_features = num_unique_word
max_words = 500
batch_size = 128
BATCH_SIZE =128
epochs = 3
num_classes=2
```

```
1416
```

4.3.2 Tokenizer

This class allows to vectorize a text corpus, by turning each text into either a sequence of integers (each integer being the index of a token in a dictionary) or into a vector where the coefficient for each token could be binary, based on word count, based on tf-idf (Text Preprocessing - Keras Documentation. (2020)).

We use tokenizer's fit_on_texts method to generate a token dictionary, and the total number of words in the dictionary is 74064. We can get a dictionary of words and its corresponding occurrences through the tokenizer.word_counts attribute. For example, the "stuff" appears 1174 times, and the "going" appears 4101 times. The tokenizer.texts_to_sequences method can help us to show the reviews as the id corresponding to the word, so as to transform it into a vector. Finally, the padding method is used to fill the reviews to the same length, so that we can vectorize the information in the embedding layer of the model and input it into the LSTM. From the above, we can see that the maximum number of words for a single review is 1146, so we use the padding method to fill all reviews to the length of 1146.

```
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(list(train_text))
print(tokenizer.word_counts)
```

```
OrderedDict([('stuff', 1174), ('going', 4101), ('moment', 1112), ('mj', 35), ('started', 963), ('listening', 187), ('music', 3056), ('watc
hing', 4603), ('odd', 582), ('documentary', 953), ('watched', 2236), ('wiz', 10), ('moonwalker', 24), ('maybe', 2340), ('want', 3703), ('g
et', 9310), ('certain', 764), ('insight', 187), ('guy', 3035), ('thought', 3437), ('really', 11736), ('cool', 971), ('eighties', 101), ('m
ake', 8023), ('mind', 1995), ('whether', 856), ('guilty', 198), ('innocent', 416), ('part', 4042), ('biography', 82), ('feature', 791),
('film', 40147), ('remember', 1702), ('see', 11475), ('cinema', 1491), ('originally', 290), ('released', 986), ('subtle', 434), ('message
s', 131), ('feeling', 1145), ('towards', 637), ('press', 127), ('also', 9156), ('obvious', 1066), ('message', 829), ('drugs', 325), ('ba
d', 9301), ('kay', 91), ('visually', 259), ('impressive', 500), ('course', 2506), ('michael', 1333), ('jackson', 340), ('unless', 675),
('remotely', 189), ('like', 20274), ('anyway', 1117), ('hate', 789), ('find', 4131), ('boring', 1809), ('may', 3386), ('call', 923), ('ego
tist', 5), ('consenting', 5), ('making', 2961), ('movie', 44031), ('fans', 1421), ('would', 12436), ('say', 5395), ('made', 8362), ('tru
e', 2333), ('nice', 2012), ('actual', 793), ('bit', 3054), ('finally', 1536), ('starts', 1220), ('minutes', 2952), ('excluding', 14), ('sm
ooth', 128), ('criminal', 325), ('sequence', 875), ('joe', 690), ('pesci', 29), ('convincing', 538), ('psychopathic', 23), ('powerful', 62
0), ('drug', 403), ('lord', 348), ('wants', 1287), ('dead', 1881), ('beyond', 866), ('overheard', 7), ('plans', 204), ('nah', 18), ('chara
cter', 7023), ('ranted', 4), ('wanted', 1352), ('people', 9285), ('know', 6166), ('supplying', 10), ('etc', 1212), ('dunno', 24), ('hate
s', 104), ('lots', 799), ('things', 3688), ('turning', 344), ('car', 1225), ('robot', 217), ('whole', 3078), ('speed', 249), ('demon', 18
3), ('director', 4444), ('must', 3249), ('patience', 81), ('saint', 76), ('came', 1673), ('filming', 393), ('kiddy', 5), ('usually', 981),
('directors', 675), ('working', 794), ('one', 26788), ('kid', 1199), ('let', 2341), ('alone', 1061), ('bunch', 813), ('performing', 131),
('complex', 427), ('dance', 744), ('scene', 5378), ('bottom', 421), ('line', 1870), ('level', 963), ('another', 4325), ('think', 7296),
('stay', 787), ('away', 2775), ('try', 1830), ('give', 3376), ('wholesome', 45), ('ironically', 123), ('bestest', 2), ('buddy', 258), ('gi
rl', 2853), ('truly', 1743), ('talented', 586), ('ever', 5995), ('grace', 328), ('planet', 502), ('well', 10662), ('attention', 906), ('ga
```

```
X_train = tokenizer.texts_to_sequences(train_text)
for i in range(3):
    print(X_train[i])
X_train = sequence.pad_sequences(X_train, maxlen=1416)
X_train, X_test, y_train, y_test = train_test_split(X_train, y, test_size = 0.05, random_state = 100)
```

```
[404, 70, 419, 8815, 506, 2456, 115, 54, 873, 516, 178, 18686, 178, 11242, 165, 78, 14, 662, 2457, 117, 92, 10, 499, 4074, 165, 22, 210, 58
1, 2333, 1194, 11242, 71, 4826, 71, 635, 2, 253, 70, 11, 302, 1663, 486, 1144, 3265, 8815, 411, 793, 3342, 17, 441, 600, 1500, 15, 4424, 185
1, 998, 146, 342, 1442, 743, 2424, 4, 8815, 418, 70, 637, 69, 237, 94, 541, 8815, 26055, 26056, 120, 1, 8815, 323, 8, 47, 20, 323, 167, 10,
207, 633, 635, 2, 116, 291, 382, 121, 15535, 3315, 1501, 574, 734, 10013, 923, 11578, 822, 1239, 1408, 360, 8815, 221, 15, 576, 8815, 22224,
2274, 13426, 734, 10013, 27, 28606, 340, 16, 41, 18687, 1500, 388, 11243, 165, 3962, 8815, 115, 627, 499, 79, 4, 8815, 1430, 380, 2163, 114,
1919, 2503, 574, 17, 60, 100, 4875, 5100, 260, 1268, 26057, 15, 574, 493, 744, 637, 631, 3, 394, 164, 446, 114, 615, 3266, 1160, 684, 48, 11
75, 224, 1, 16, 4, 8815, 3, 507, 62, 25, 16, 640, 133, 231, 95, 7426, 600, 3439, 8815, 37248, 1864, 1, 128, 342, 1442, 247, 3, 865, 16, 42,
1487, 997, 2333, 12, 549, 386, 717, 6920, 12, 41, 16, 158, 362, 4392, 3388, 41, 87, 225, 438, 207, 254, 117, 3, 18688, 18689, 316, 1356]
[232, 203, 3048, 3565, 7116, 317, 2, 405, 153, 19, 634, 10967, 11898, 8816, 1653, 1035, 3494, 232, 154, 314, 7116, 2701, 178, 2, 2349, 87, 1
111, 582, 217, 2219, 149, 73, 160, 626, 1035, 2882, 194, 642, 3316, 3464, 3869, 154, 405, 180, 155, 158, 79, 1, 19718, 2177, 1251, 68, 6828,
170, 281, 811, 1, 532, 10968, 4, 2003, 16, 36, 881, 1251, 376, 634, 314, 7116, 159, 2564, 1653, 1035, 3494, 232, 511, 143, 317, 20, 623, 462
8, 1251, 8974, 5471]
[2, 382, 2818, 4393, 3730, 598, 2184, 17788, 518, 4313, 9355, 1328, 837, 1012, 37249, 9355, 1435, 624, 46199, 11244, 4, 37250, 1328, 3102, 1
1245, 3, 714, 18690, 13004, 9550, 4592, 32129, 3542, 5590, 374, 2305, 86, 193, 19719, 8332, 2869, 1443, 924, 630, 13891, 6048, 4954, 279, 97
78, 8817, 3542, 1920, 8500, 2350, 10474, 1454, 2366, 2046, 2827, 2911, 899, 1569, 1217, 1208, 37251, 1791, 1416, 2366, 1362, 12248, 37252, 3
465, 1396, 37253, 409, 832, 22225, 46200, 22226, 146, 53, 201, 256, 6135, 16218, 55, 923, 1301, 1208, 13891, 6048, 4954, 459, 384, 16219, 40
9, 3, 714, 18690, 13004, 3900, 687, 9550, 1656, 413, 8818, 1308, 1, 1382, 6301, 627, 399, 455, 15536, 991, 4314, 5218, 244, 2425, 22226, 82
4, 1350, 198, 183, 9, 1403, 968, 8023, 571, 1750, 77, 237, 1301, 1208, 1999, 20, 1064, 18691, 191, 354, 2143, 17789, 233, 55, 1684, 12249, 5
048, 1409, 1472, 59, 95, 19720, 1578, 233, 15537, 13427, 481, 2552, 8666, 13005, 3227, 6218, 505, 344, 2305, 48, 266, 229, 672, 2030, 953, 1
109, 49, 30, 22226, 32129, 891, 9550, 439, 1078, 22227, 5591, 2064, 466, 4556, 182, 8975, 3300, 13, 44, 13428, 10724, 28607, 1975, 15538, 37
31, 3072, 16986, 7427, 1110, 304, 906, 1602, 262, 759, 392, 578, 2674, 2656, 187, 32, 645, 808, 30, 2674, 2251, 60, 493, 631, 551, 23948, 26
35, 1123, 2046, 534, 32, 281, 1875, 302, 43, 8501, 1671, 23949, 23950, 4356, 32130, 535, 700, 1175, 5310]
```

4.3.3 Keras

In the experiment, we need to use Keras API to help us build the classifier model. There are three different layers in our classifier model.

- **Embedding Layer**. Keras provides an embedding layer, which is helpful for the neural network of text data. It requires the input data to be the integer, and each word could be represented by a unique integer. The embedding layer will be initialized with random weights and embed all words in the training dataset.

- **Long Short-Term Memory Layer**. In this experiment, we can use the keras.layers.lstm method to directly construct the LSTM layer. The implementation principle of the specific LSTM layer has been introduced in the previous section.

- **Core Layers**. Dense layer implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bias is True). We also apply Dropout layer to the input. Dropout layer consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting (Core Layers - Keras Documentation. (2020)).

In this experiment, we construct a tf.keras.Sequential model and start with the embedding layer. The embedding layer stores a vector for each word. When it's called, it converts the sequence of word indices into a sequence of vectors. At the same time, these vectors are trainable. After they are trained (on enough data), words with similar meanings usually have similar vectors.

What's more, this index lookup method is more efficient than the equivalent operation of transferring one-hot encoding vector through the Dense layer. We also use tf.keras.layers.Bidirectional wrapper with LSTM layer together. It can propagate the input forward and backward through the LSTM layer and then connect the output, which helps LSTM to learn remote dependencies.

After the model is built, we randomly select train_data and test_data according to the ratio of 0.05 through the train_test_split method, that is, 23750 data in train_data and

1250 data in test_data. And then we put the dataset into the model for training, the training details as follows:

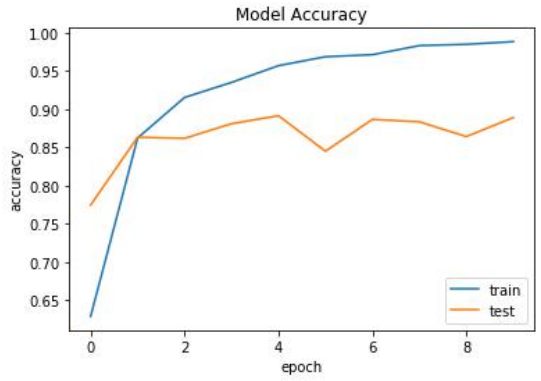
```
DROPOUT_RATE = 0.5
HIDDEN_UNITS = 64
model2=Sequential()
model2.add(layers.Embedding(max_features,100,mask_zero=True))
model2.add(layers.Bidirectional(layers.LSTM(64, return_sequences=False)))
model2.add(layers.Dropout(DROPOUT_RATE))
model2.add(layers.Dense(num_classes,activation='softmax'))
model2.compile(loss='categorical_crossentropy',optimizer=keras.optimizers.Adam(lr=1e-4),metrics=['accuracy'])
model2.summary()
history2=model2.fit(X_train, y_train, validation_data=(X_test, y_test),epochs=10, batch_size=BATCH_SIZE, verbose=1)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 100)         7406400
_________________________________________________________________
bidirectional (Bidirectional (None, 128)               84480
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 2)                 258
=================================================================
Total params: 7,491,138
Trainable params: 7,491,138
Non-trainable params: 0
_________________________________________________________________
Train on 23750 samples, validate on 1250 samples
Epoch 1/10
23750/23750 [==============================] - 12032s 507ms/sample - loss: 0.6711 - accuracy: 0.6288 - val_loss: 0.5019 - val_accuracy: 0.7744
Epoch 2/10
23750/23750 [==============================] - 8511s 358ms/sample - loss: 0.3636 - accuracy: 0.8621 - val_loss: 0.3621 - val_accuracy: 0.8632
Epoch 3/10
23750/23750 [==============================] - 8199s 345ms/sample - loss: 0.2422 - accuracy: 0.9151 - val_loss: 0.3337 - val_accuracy: 0.8616
Epoch 4/10
23750/23750 [==============================] - 8233s 347ms/sample - loss: 0.1996 - accuracy: 0.9348 - val_loss: 0.3310 - val_accuracy: 0.8808
Epoch 5/10
23750/23750 [==============================] - 8002s 337ms/sample - loss: 0.1350 - accuracy: 0.9567 - val_loss: 0.3179 - val_accuracy: 0.8912
Epoch 6/10
23750/23750 [==============================] - 7638s 322ms/sample - loss: 0.1067 - accuracy: 0.9683 - val_loss: 0.4020 - val_accuracy: 0.8448
Epoch 7/10
23750/23750 [==============================] - 7687s 324ms/sample - loss: 0.0975 - accuracy: 0.9710 - val_loss: 0.3580 - val_accuracy: 0.8864
Epoch 8/10
23750/23750 [==============================] - 7828s 330ms/sample - loss: 0.0616 - accuracy: 0.9829 - val_loss: 0.3858 - val_accuracy: 0.8832
Epoch 9/10
23750/23750 [==============================] - 7361s 310ms/sample - loss: 0.0561 - accuracy: 0.9846 - val_loss: 0.4790 - val_accuracy: 0.8640
Epoch 10/10
23750/23750 [==============================] - 7267s 306ms/sample - loss: 0.0445 - accuracy: 0.9881 - val_loss: 0.4057 - val_accuracy: 0.8888
```

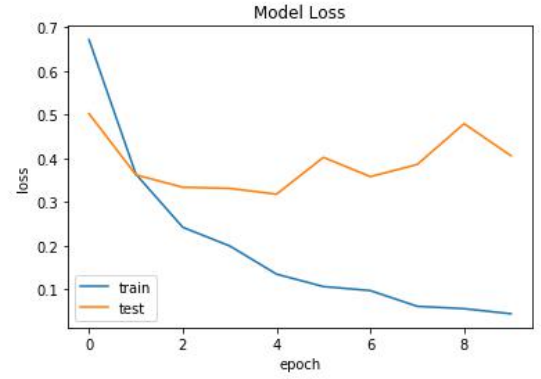
From the figure below, we can see that the model gradually improves its accuracy through continuous training, from 0.7744 to 0.8912, and then gradually stabilizes at about 0.88. Finally, we can get the highest accuracy of the model is 0.8912. We also show the changes in accuracy and loss of the model during training through the following

figure. From the left figure, we can see that for the accuracy rate, the accuracy of the training dataset has been continuously rising, and finally stabilized at about 0.98. The accuracy of the test dataset increases from 0.77 firstly, and then gradually stabilizes at about 0.88. From the right figure, we can see that for the loss rate, the loss of the training dataset gradually decreases from 0.67 and eventually stabilizes at about 0.05. The loss of the test dataset drops from 0.5 and then gradually stabilizes at about 0.4.

```
fig = plt.figure()
plt.plot(history2.history['accuracy'])
plt.plot(history2.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower right')
plt.show()
```




```
fig = plt.figure()
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower left')
plt.show()
```



4.3.4 Model Testing

We can also test the classifier model with some simple sentences. For example, we can input some sentences with known emotions into the model such as "This movie is excellent.", "I love this movie." and "It is boring." We can get the emotion prediction probability of the model, compare these emotion probabilities, and then convert the probability to the corresponding emotional result (positive or negative). From the figure below, we can see that the classifier model predicts that the sentiments of these three

sentences are positive, positive, and negative, respectively. It is exactly the same as the emotional results we know, so we think that the model is relatively accurate and stable.

```
x_test = ['This movie is excellent.','I love this movie.','It is boring.']
x_test1 = tokenizer.texts_to_sequences(x_test)
x_test2 = sequence.pad_sequences(x_test1, maxlen = 500)
labels = [int(round(x[1])) for x in model2.predict(x_test2) ]
label2word = {1:'Positive', 0:'Negative'}
for i in range(len(x_test)):
    print('{}    {}'.format(label2word[labels[i]], x_test[i]))
```

```
Positive   This movie is excellent.
Positive   I love this movie.
Negative   It is boring.
```

4.3.5 Various Indicators

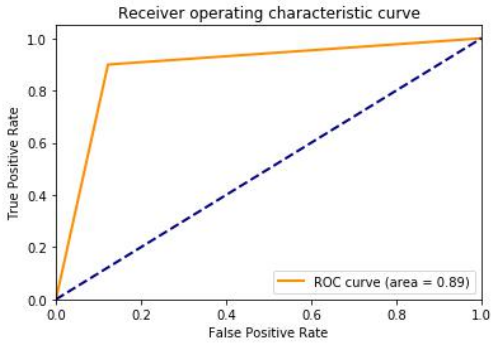
Through our learning in the Naïve Bayes experiment, we can also test the following indicators on the LSTM model. From the figure, we can draw that the average accuracy of the LSTM model is 0.8888. The ROC curve area is 0.8886. For the negative class, the precision is 0.89, the recall is 0.88, and the F1 threshold is 0.89. For the positive class, the precision is 0.88, the recall is 0.90, and the F1 threshold is 0.89. A comprehensive comparison of the two classes shows that the precision, recall, and F1 thresholds of this model are all 0.89.

```
pred=model2.predict(X_test)
pred1=[]
for i in range(len(pred)):
    pred[i][0]=int(round(pred[i][0]))
    pred[i][1]=int(round(pred[i][1]))
    if(pred[i][0]<pred[i][1]):
        pred1.append(1)
    else:
        pred1.append(0)
y_test1=[]
for i in range(len(y_test)):
    if(y_test[i][0]<y_test[i][1]):
        y_test1.append(1)
    else:
        y_test1.append(0)
```

```
# accuracy of model
accuracy_score(y_test1, pred1)
```

```
0.8888
```

```
fpr, tpr, thresholds = metrics.roc_curve(y_test1, pred1, pos_label=1)
#ROC curve
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',lw=lw, label='ROC curve (area = %0.2f)' % accuracy_score(y_test1, pred1))
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='—')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic curve')
plt.legend(loc="lower right")
plt.show()
```




```
# auc area
metrics.auc(fpr, tpr)
```

```
0.8885899698064695
```

```
print(classification_report(y_test1, pred1))
```

```
              precision    recall  f1-score   support

           0       0.89      0.88      0.89       613
           1       0.88      0.90      0.89       637

    accuracy                           0.89      1250
   macro avg       0.89      0.89      0.89      1250
weighted avg       0.89      0.89      0.89      1250
```

5. Discussion

In the previous two experiments, we constructed the Naïve Bayes model and the LSTM model separately, and trained, tested, and verified them, and finally obtained a series of indicator results. The indicator results of the two experiments are now organized into the following table:

	Average Accuracy	Validation Accuracy	ROC Curve Area	Precission	Recall	F1-Score
Naïve Bayes	0. 8498	0. 8424	0. 8501	0. 85	0. 85	0. 85
LSTM	0. 8888	0. 8912	0. 8886	0. 89	0. 89	0. 89

From the table, we can compare the Naïve Bayes model and the LSTM model. For the average accuracy and validation accuracy, the accuracy of the LSTM model is 0.8888 and 0.8912, which are higher than the Naïve Bayes model of 0.8498 and 0.8424. For the ROC curve, we obviously find that the AUC area of the LSTM model is 0.8886, which is much larger than the Naïve Bayes model of 0.8501. In the sentiment analysis experiment of movie reviews, precision is more important than F1-score, and recall is the least important. It can be seen from the above figure that the precision, recall, and F1 thresholds of the Naïve Bayes model are 0.85, while the precision, recall, and F1 thresholds of the LSTM model are 0.89. Whether it is comparing all three indicators or individual precision, the LSTM model is better than the Naïve Bayes model. In summary, we can think that the LSTM model is far superior to the Naïve Bayes model in all indicators. When we build a web later, we will import these two models, so that users can more clearly and accurately conduct sentiment analysis of reviews and compare their analysis results.

6. Web Development

Users can input the reviews they want to analyze through our front-end interface and select the corresponding models. The front-end interface will make judgments on the users' choices and transfer the contents to the corresponding model. The model will perform sentiment analysis users' contents, and feedback the corresponding sentiment probability to the front-end interface. So users can see the corresponding analysis results intuitively and clearly on the interface.

6.1 Flask

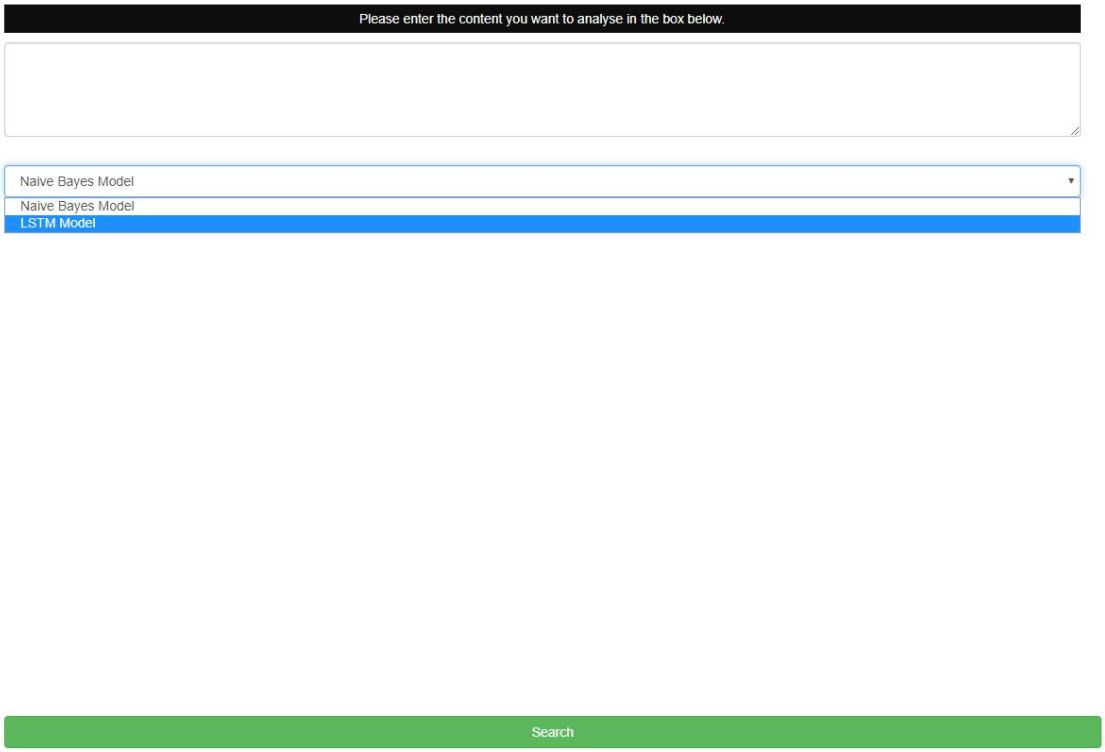
Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It offers suggestions but doesn't enforce any dependencies or project layout. It is up to the developer to choose the tools and libraries they want to use. There are many extensions provided by the community that make adding new functionality easy (Flask. (2020)).

We will write a new python file in the python environment. In this file, we mainly import the Naïve Bayes and LSTM models that we have previously built into the web by calling the flask library, connect it to the front-end file index.html and implement the web architecture. See Appendix C for the specific code of the python file and index.html file.

6.2 Web

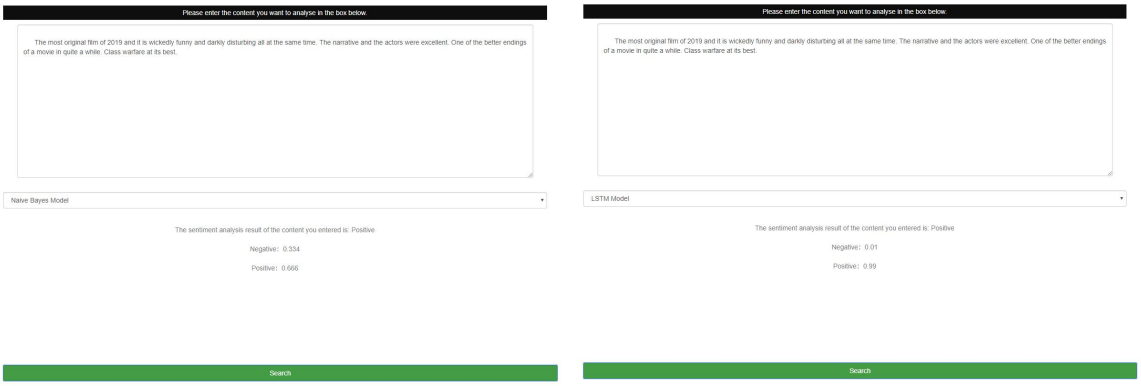
On the web, we can see users can enter the content they want to analyze in the box, and select the model they want to analyze, and click the search button to get the sentiment

analysis results and probability of the corresponding model for the content. As shown below:

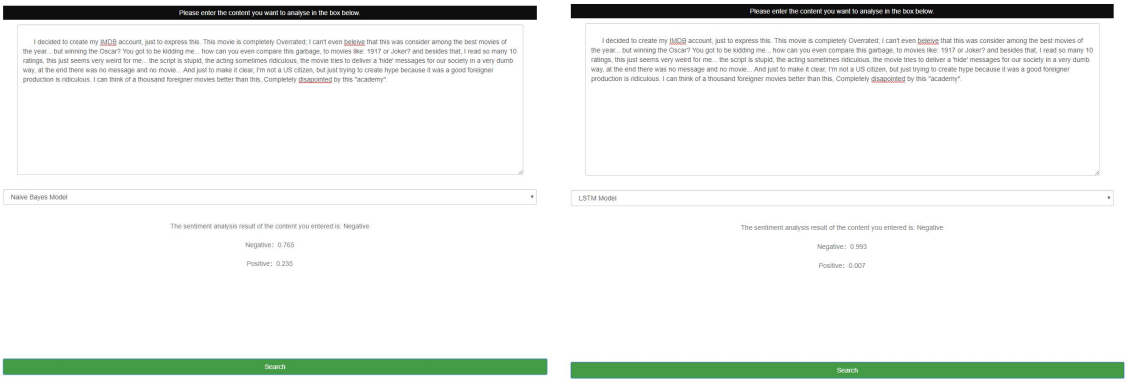


Here we can test some specific examples. We found three user reviews with a rating of 1, 5, 10 in the reviews of the movie Parasite.

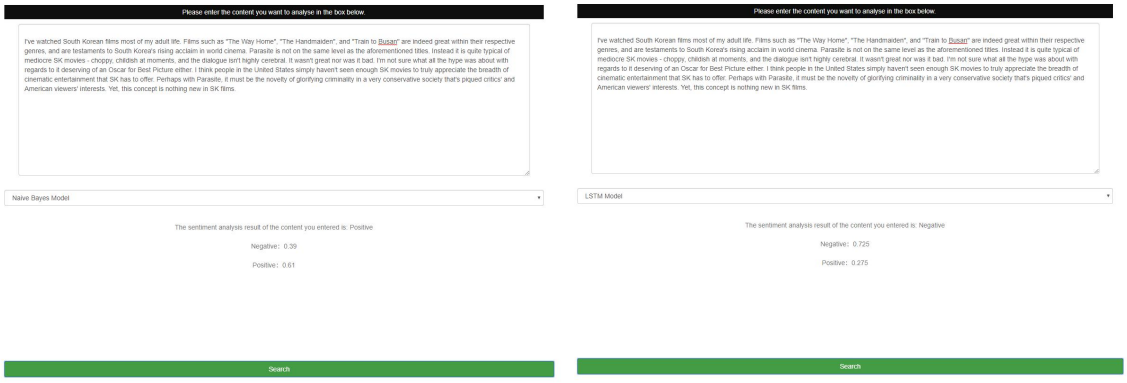
Firstly, we input a 1 point review, and we can find that the sentiment analysis results made by the Naïve Bayes and LSTM models are both positive, but the probabilities are different. For the Naïve Bayes model, the positive probability of this review is 0.666, and the negative probability is 0.334. For the LSTM model, the positive probability of this review is 0.99, and the negative probability is 0.01.



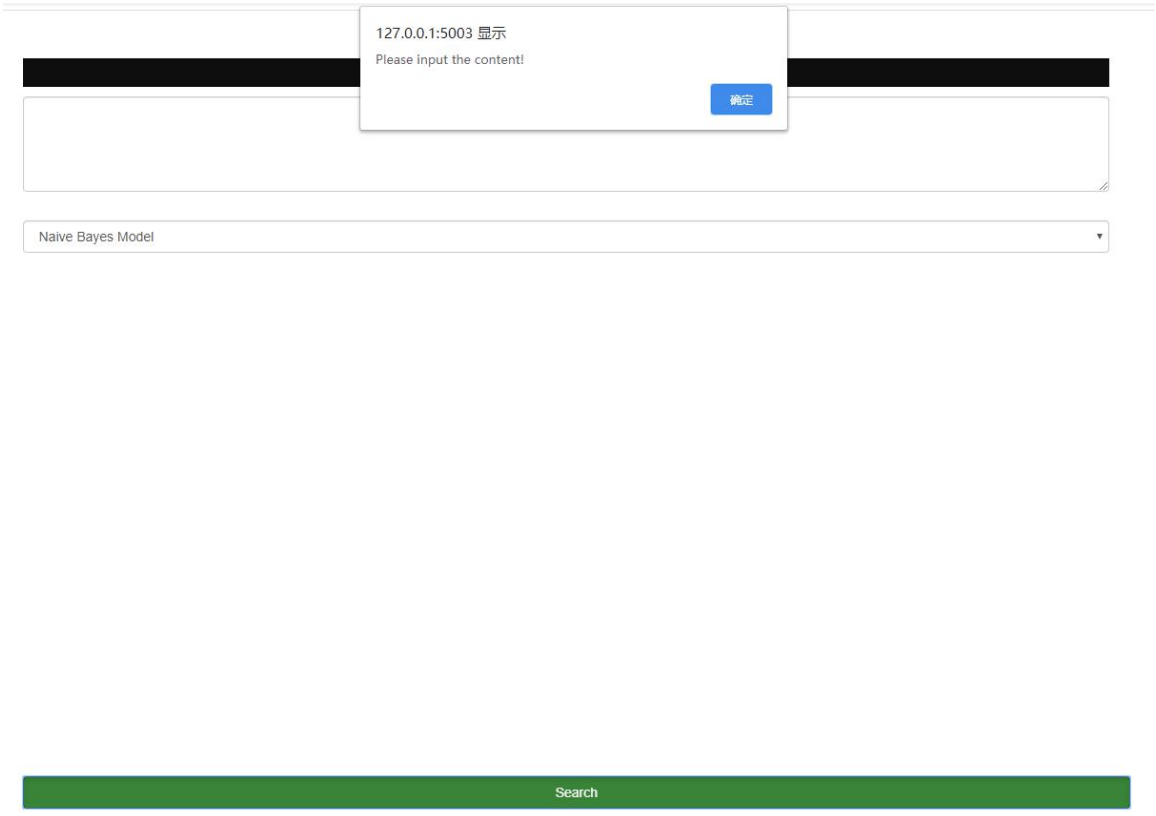
When we input a 10 point review, we can find that the sentiment analysis results made by the Naïve Bayes and LSTM models are both negative. For the Naïve Bayes model, the positive probability of this review is 0.235, and the negative probability is 0.765. For the LSTM model, the positive probability of this review is 0.007, and the negative probability is 0.993.



Finally, we input a 5 point review, and we find that the sentiment analysis results of two models are different. For the Naïve Bayes model, this review is positive, its positive probability is 0.61, and the negative probability is 0.39. For the LSTM model, this review is negative, its positive probability is 0.275, and the negative probability is 0.725.



Additionally, we also made some user-friendly settings and reminders for the web. For example, when the content input by the user is empty, we will prompt the user to input the content through a prompt box.



7. Conclusion

The research on the sentiment analysis of online movie reviews has a significant meaning for the types and proportions of the movie market. Through our research, we can know that it is feasible to combine sentiment analysis of the content with machine learning algorithms, and we can construct an algorithm model that reduces workload and improves accuracy. Here, our main task of this paper is to complete the sentiment analysis of online movie reviews. We have selected two currently popular emotion classification models. By comparing and in-depth analysis of the classification results of the two classification models, we summarize the advantages and disadvantages of the two machine learning algorithms and find a model that is more suitable for the sentiment analysis of online movie reviews. Of course, our research is not perfect because of various reasons, and it still has room for improvement. In future research, we will further improve the above developments and make the study more abundant and more valuable.

8. Bibliography

Global Regional Internet Penetration Rate 2019 | Statista. (2019). Retrieved 9 November 2019, from https://www.statista.com/statistics/269329/penetration-rate-of-the-internet-by-region/

Picard, R. (1997). Affective computing. MIT Press.

Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up?. Proceedings Of The ACL-02 Conference On Empirical Methods In Natural Language Processing - EMNLP '02. doi: 10.3115/1118693.1118704

Whitelaw, C., Garg, N., & Argamon, S. (2005). Using appraisal groups for sentiment analysis. Proceedings Of The 14Th ACM International Conference On Information And Knowledge Management - CIKM '05. doi: 10.1145/1099554.1099714

Boiy, E., & Moens, M. (2008). A machine learning approach to sentiment analysis in multilingual Web texts. Information Retrieval, 12(5), 526-558. doi: 10.1007/s10791-008-9070-z

Turney, P. (2001). Thumbs up or thumbs down?. Proceedings Of The 40Th Annual Meeting On Association For Computational Linguistics - ACL '02. doi: 10.3115/1073083.1073153

Teng, Z., Vo, D., & Zhang, Y. (2016). Context-Sensitive Lexicon Features for Neural Sentiment Analysis. Proceedings Of The 2016 Conference On Empirical Methods In Natural Language Processing. doi: 10.18653/v1/d16-1169

Qian Q., Huang, M., Lei, J., & Zhu, X. (2016). Linguistically regularized lstms for sentiment classification. arXiv preprint arXiv:1611.03949

Cliche, M. (2017). Twitter Sentiment Analysis with CNNs and LSTMs. arXiv preprint arXiv:1704.06125

Pong-inwong, C., & Songpan, W. (2018). Sentiment analysis in teaching evaluations using sentiment phrase pattern matching (SPPM) based on association mining. International Journal Of Machine Learning And Cybernetics, 10(8), 2177-2186. doi: 10.1007/s13042-018-0800-2

Shen, C., & J, J. (2018). Text Sentiment Classification Algorithm Based on Double Channel Convolutional Neural Network. Moshi Shibie yu Rengong Zhineng/Pattern Recognition and Artificial Intelligence, 31. 158-166. doi: 10.16451/j.cnki.issn1003-6059.201802007

Dave, K., Lawrence, S., & Pennock, D. (2003). Mining the peanut gallery. Proceedings Of The Twelfth International Conference On World Wide Web - WWW '03. doi: 10.1145/775152.775226

Gamon, M., Aue, A., Corston-Oliver, S., & Ringger, E. (2005). Pulse: Mining Customer Opinions from Free Text. Lecture Notes In Computer Science, 121-132. doi: 10.1007/11552253_12

T. Wilson, P. Hoffmann, S. Somasundaran, J. Kessler, J. Wiebe, Y. Choi, C. Cardie, E. Riloff & S. Patwardhan. (2005). OpinionFinder: A system for subjectivity analysis. Demonstration Description in Conference on Empirical Methods in Natural Language Processing.

Jeonghee Yi., & Niblack, W. (2005). Sentiment Mining in WebFountain. 21St International Conference On Data Engineering (ICDE'05). doi: 10.1109/icde.2005.132

Liu, B., Hu, M., & Cheng, J. (2005). Opinion observer. Proceedings Of The 14Th International Conference On World Wide Web - WWW '05. doi: 10.1145/1060745.1060797

Bosco, C., Patti, V., & Bolioli, A. (2013). Developing Corpora for Sentiment Analysis: The Case of Irony and Senti-TUT. IEEE Intelligent Systems, 28(2), 55-63. doi: 10.1109/mis.2013.28

Tf-idf :: A Single-Page Tutorial - Information Retrieval and Text Mining. (2020). Retrieved 11 March 2020, from http://www.tfidf.com/

Removing stop words with NLTK in Python - GeeksforGeeks. (2020). Retrieved 11 March 2020, from https://www.geeksforgeeks.org/removing-stop-words-nltk-python/

1.9. Naïve Bayes — scikit-learn 0.22.2 documentation. (2020). Retrieved 11 March 2020, from https://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes

Cross-validation (statistics). (2020). Retrieved 11 March 2020, from https://en.wikipedia.org/wiki/Cross-validation_(statistics)

Receiver operating characteristic. (2020). Retrieved 11 March 2020, from https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735-1780. doi: 10.1162/neco.1997.9.8.1735

Text Preprocessing - Keras Documentation. (2020). Retrieved 21 March 2020, from https://keras.io/preprocessing/text/

Core Layers - Keras Documentation. (2020). Retrieved 21 March 2020, from https://keras.io/layers/core/

Flask. (2020). Retrieved 1 April 2020, from https://palletsprojects.com/p/flask/

Appendix A:

The complete Naïve Bayes model code:

Naïve Bayes Modeling

In [1]:
```
#import pandas, numpy, sklearn, Nltk and matplotlib package
import pandas as pd
import numpy as np
from sklearn.naive_bayes import MultinomialNB
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer,TfidfTransformer,TfidfVectorizer
import nltk
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection   import train_test_split
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer as TFIDF
from sklearn.metrics import accuracy_score,confusion_matrix,precision_recall_curve,average_precision_score
from sklearn import metrics
from pylab import *
import numpy as np
from sklearn.metrics import roc_curve, auc
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.externals import joblib
import re
from sklearn.metrics import classification_report
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\寿司\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[1]:
```
True
```

In [2]:
```
#load dataset
train = pd.read_csv('word2vec-nlp-tutorial/labeledTrainData.tsv', delimiter="\t")
train.head()
```

Out[2]:

	id	sentiment	review
0	5814_8	1	With all this stuff going down at the moment w...
1	2381_9	1	\The Classic War of the Worlds\" by Timothy Hi...
2	7759_3	0	The film starts with a manager (Nicholas Bell)...
3	3630_4	0	It must be assumed that those who praised this...
4	9495_8	1	Superbly trashy and wondrously unpretentious 8...

In [3]:
```
def review_to_wordlist(review):
    #Only save English words
    review_text = re.sub("[^a-zA-Z]"," ", review)
    #to lower case
    words = review_text.lower()
    return(words)

# Separate the reviews and sentiments
# 1. extract the sentiments
y_train_1 = train['sentiment']
# 2. extract the reviews
train_data = []
for review in train['review']:
    train_data.append(review_to_wordlist(review))
# 3. transfer to numpy array
train_data = np.array(train_data)
```

In [4]:
```
train_data
```

Out[4]:
```
array(['with all this stuff going down at the moment with mj i ve started listening to his music  watching the odd documentary here and ther
e  watched the wiz and watched moonwalker again  maybe i just want to get a certain insight into this guy who i thought was really cool in t
he eighties just to maybe make up my mind whether he is guilty or innocent  moonwalker is part biography  part feature film which i remember
going to see at the cinema when it was originally released  some of it has subtle messages about mj s feeling towards the press and also the
obvious message of drugs are bad m kay  br    br  visually impressive but of course this is all about michael jackson so unless you remotel
y like mj in anyway then you are going to hate this and find it boring  some may call mj an egotist for consenting to the making of this mov
ie but mj and most of his fans would say that he made it for the fans which if true is really nice of him  br    br  the actual feature fil
m bit when it finally starts is only on for    minutes or so excluding the smooth criminal sequence and joe pesci is convincing as a psychop
athic all powerful drug lord  why he wants mj dead so bad is beyond me  because mj overheard his plans  nah  joe pesci s character ranted th
at he wanted people to know it is he who is supplying drugs etc so i dunno  maybe he just hates mj s music  br    br  lots of cool things i
n this like mj turning into a car and a robot and the whole speed demon sequence  also  the director must have had the patience of a saint w
hen it came to filming the kiddy bad sequence as usually directors hate working with one kid let alone a whole bunch of them performing a co
mplex dance scene  br    br  bottom line  this movie is for people who like mj on one level or another  which i think is most people  if n
ot  then stay away  it does try and give off a wholesome message and ironically mj s bestest buddy in this movie is a girl  michael jackson
is truly one of the most talented people ever to grace this planet but is he guilty  well  with all the attention i ve gave this subject
hmmm well i don t know because people can be different behind closed doors  i know this for a fact  he is either an extremely nice but stupi
d guy or one of the most sickest liars  i hope he is not the latter ',
```

In [5]:
```
y_train_1
```

Out[5]:
```
0        1
1        1
2        0
3        0
4        1
5        1
6        0
7        0
8        0
9        1
10       0
11       1
12       1
13       0
14       0
15       0
16       0
17       0
18       1
19       1
20       1
21       1
22       1
23       0
24       0
25       1
26       0
27       0
28       0
29       0
        ..
24970    1
24971    1
24972    1
24973    0
24974    1
24975    1
24976    0
24977    1
24978    1
24979    1
24980    1
24981    1
24982    0
24983    0
24984    0
24985    0
24986    1
24987    1
24988    1
24989    1
24990    1
24991    0
24992    0
24993    0
24994    0
24995    0
24996    0
24997    0
24998    0
24999    1
Name: sentiment, Length: 25000, dtype: int64
```

In [6]:
```
tfidf = TfidfVectorizer(ngram_range=(1, 2), min_df=0.01, use_idf=1,smooth_idf=1, stop_words = 'english') # remove English stop words
data_train_count = tfidf.fit_transform(train_data)
```

In [7]:
```
words = tfidf.get_feature_names()
print(words[1398])
print(words[609])
print(words[950])
```

```
stuff
going
moment
```

In [8]:
```
print(data_train_count)
```

```
  (0, 1398)      0.07359370363384517
  (0, 609)       0.15836051874700902
  (0, 950)       0.07480337141431258
  (0, 1546)      0.09991275705206315
  (0, 1370)      0.07651748600849363
  (0, 982)       0.11911861384547107
  (0, 1586)      0.05061903444777294
  (0, 1022)      0.08615487797747001
  (0, 381)       0.08259726086604724
  (0, 1584)      0.12453812750958633
```

In [9]:
```
X_train, X_test, y_train, y_test = train_test_split(data_train_count, y_train_1, test_size = 0.2, random_state = 0)
```

In [10]:
```
#multinomialNB model
clf = MultinomialNB()
clf.fit(X_train, y_train)
pred = clf.predict(X_test)
```

In [11]:
```
#score of test dataset
clf.score(X_test, y_test)
```

Out[11]: 0.8498

In [12]:
```
#score of train dataset
clf.score(X_train, y_train)
```

Out[12]: 0.85035

In [13]:
```
#probability estimates for the test
clf.predict_proba(X_test)
```

Out[13]:
```
array([[0.48816433, 0.51183567],
       [0.45184684, 0.54815316],
       [0.27062664, 0.72937336],
       ...,
       [0.40218761, 0.59781239],
       [0.84719866, 0.15280134],
       [0.2512578 , 0.7487422 ]])
```

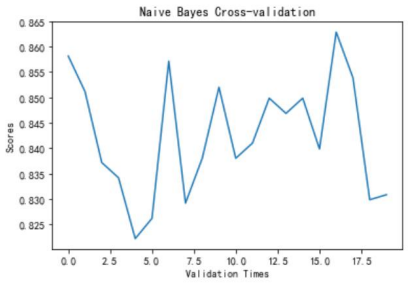
In [14]:
```
# accuracy of model
accuracy_score(y_test, pred)
```

Out[14]: 0.8498

In [15]:
```
times = 20
scores = cross_val_score(clf, X_train, y_train, cv=times)
```

In [16]:
```
mpl.rcParams['font.sans-serif'] = ['SimHei']
plt.title('Naive Bayes Cross-validation')
plt.xlabel("Validation Times")
plt.ylabel("Scores")
plt.plot(range(times), scores)
```

Out[16]: [<matplotlib.lines.Line2D at 0x25334677208>]



In [17]:
```
avgscore = np.mean(scores)
print(avgscore)
```

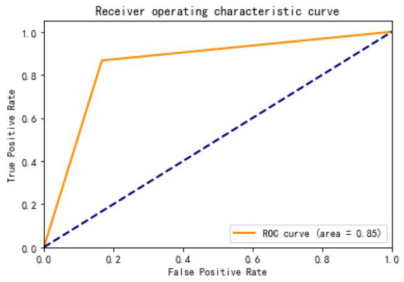
```
0.8424024239524239
```

In [18]:
```
#confusion matrix
confusion_matrix(y_test, pred, labels=None, sample_weight=None)
```

Out[18]:
```
array([[2124,  424],
       [ 327, 2125]], dtype=int64)
```

In [19]:
```
fpr, tpr, thresholds = metrics.roc_curve(y_test, pred, pos_label=1)
```

In [20]:
```
#ROC curve
import matplotlib.pyplot as plt
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',lw=lw, label='ROC curve (area = %0.2f)' % accuracy_score(y_test, pred))
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic curve')
plt.legend(loc="lower right")
plt.show()
```



In [21]:
```
# auc area
metrics.auc(fpr, tpr)
```

Out[21]: 0.8501172272146404

In [22]:
```
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.83      0.85      2548
           1       0.83      0.87      0.85      2452

   micro avg       0.85      0.85      0.85      5000
   macro avg       0.85      0.85      0.85      5000
weighted avg       0.85      0.85      0.85      5000
```

In [23]:
```
#save model
joblib.dump(tfidf,'tf_model.m')
joblib.dump(clf,'clf_model.m')
```

Out[23]: ['clf_model.m']

Appendix B:

The complete LSTM model code:

LSTM Modeling

In [21]:
```
from tensorflow.python.keras import Sequential
from tensorflow.python.keras.preprocessing import sequence
from tensorflow.keras import layers
from tensorflow import keras
from tensorflow.python.keras.utils import multi_gpu_model
from tensorflow.keras.models import load_model
from keras.preprocessing.text import Tokenizer
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score,confusion_matrix,precision_recall_curve,average_precision_score
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import classification_report
import os
import numpy as np
import pandas as pd
import nltk
from nltk.tokenize import word_tokenize
from nltk import FreqDist
from nltk.stem import SnowballStemmer,WordNetLemmatizer
from string import punctuation
import re
import pickle
import numpy
import matplotlib.pyplot as plt
from bs4 import BeautifulSoup
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from tensorflow.keras.preprocessing import sequence,text
stemmer=SnowballStemmer('english')
lemma=WordNetLemmatizer()
nltk.download('stopwords')
wordnet_lemmatizer = WordNetLemmatizer()
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\62531\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [22]:
```
train = pd.read_csv('word2vec-nlp-tutorial/labeledTrainData.tsv', delimiter="\t")
train.head()
```

Out[22]:

	id	sentiment	review
0	5814_8	1	With all this stuff going down at the moment w...
1	2381_9	1	\The Classic War of the Worlds\" by Timothy Hi...
2	7759_3	0	The film starts with a manager (Nicholas Bell)...
3	3630_4	0	It must be assumed that those who praised this...
4	9495_8	1	Superbly trashy and wondrously unpretentious 8...

In [23]:
```
def review_to_words( raw_review ):
    #去掉html标签
    review_text = BeautifulSoup(raw_review).get_text()
    #移除特殊字符
    letters_only = re.sub("[^a-zA-Z]", " ", review_text)
    #  全部转化为小写
    words = letters_only.lower().split()
    # 去掉停词
    stops = set(stopwords.words("english"))
    meaningful_words = [w for w in words if not w in stops]
    # 转语义
    for word in meaningful_words:
        word = wordnet_lemmatizer.lemmatize(word,'v')
    # 拼接成文字段落
    return( " ".join( meaningful_words))
```

In [24]:
```
#清除所有数据集的标签，并且转义
train['Phrase_clean_review'] = train.apply(lambda x: review_to_words(x['review']), axis=1)
train.head()
```

Out[24]:

	id	sentiment	review	Phrase_clean_review
0	5814_8	1	With all this stuff going down at the moment w...	stuff going moment mj started listening music ...
1	2381_9	1	\The Classic War of the Worlds\" by Timothy Hi...	classic war worlds timothy hines entertaining ...
2	7759_3	0	The film starts with a manager (Nicholas Bell)...	film starts manager nicholas bell giving welco...
3	3630_4	0	It must be assumed that those who praised this...	must assumed praised film greatest filmed oper...
4	9495_8	1	Superbly trashy and wondrously unpretentious 8...	superbly trashy wondrously unpretentious explo...

In [25]:
```
train_text=train.Phrase_clean_review.values
target=train.sentiment.values
y=to_categorical(target)
all_words=' '.join(train_text)
all_words=word_tokenize(all_words)
dist=FreqDist(all_words)
for key in dist:
    print(key, dist[key])
num_unique_word=len(dist)
```

```
delegate 1
unlockables 1
schnooks 1
macdowel 1
macrae 3
seasoning 1
dashes 9
coffins 6
seens 2
myrnah 1
daisy 56
witchie 1
thepeople 1
pedicure 1
pothus 1
crappily 1
pavignano 1
invasive 2
intuition 5
```

In [26]:
```
num_unique_word
```

Out[26]: 74064

In [27]:
```
r_len=[]
for text in train_text:
    word=word_tokenize(text)
    l=len(word)
    r_len.append(l)
for i in r_len:
    print(i)
```

```
219
84
240
189
210
43
55
69
93
23
35
71
200
71
58
103
213
224
138
```

In [28]:
```
MAX_REVIEW_LEN=np.max(r_len)
print(MAX_REVIEW_LEN)
max_features = num_unique_word
max_words = 500
batch_size = 128
BATCH_SIZE =128
epochs = 3
num_classes=2
```

```
1416
```

In [29]:
```
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(list(train_text))
print(tokenizer.word_counts)
```

```
OrderedDict([('stuff', 1174), ('going', 4101), ('moment', 1112), ('mj', 35), ('started', 963), ('listening', 187), ('music', 3056), ('watc
hing', 4603), ('odd', 582), ('documentary', 953), ('watched', 2236), ('wiz', 10), ('moonwalker', 24), ('maybe', 2340), ('want', 3703), ('g
et', 9310), ('certain', 764), ('insight', 187), ('guy', 3035), ('thought', 3437), ('really', 11736), ('cool', 971), ('eighties', 101), ('m
ake', 8023), ('mind', 1995), ('whether', 856), ('guilty', 198), ('innocent', 416), ('part', 4042), ('biography', 82), ('feature', 791),
('film', 40147), ('remember', 1702), ('see', 11475), ('cinema', 1491), ('originally', 290), ('released', 986), ('subtle', 434), ('message
s', 131), ('feeling', 1145), ('towards', 637), ('press', 127), ('also', 9156), ('obvious', 1066), ('message', 829), ('drugs', 325), ('ba
d', 9301), ('kay', 91), ('visually', 259), ('impressive', 500), ('course', 2506), ('michael', 1333), ('jackson', 340), ('unless', 675),
('remotely', 189), ('like', 20274), ('anyway', 1117), ('hate', 789), ('find', 4131), ('boring', 1809), ('may', 3386), ('call', 923), ('ego
tist', 5), ('consenting', 5), ('making', 2961), ('movie', 44031), ('fans', 1421), ('would', 12436), ('say', 5395), ('made', 8362), ('tru
e', 2333), ('nice', 2012), ('actual', 793), ('bit', 3054), ('finally', 1536), ('starts', 1220), ('minutes', 2952), ('excluding', 14), ('sm
ooth', 128), ('criminal', 325), ('sequence', 875), ('joe', 690), ('pesci', 29), ('convincing', 538), ('psychopathic', 23), ('powerful', 62
0), ('drug', 403), ('lord', 348), ('wants', 1287), ('dead', 1881), ('beyond', 866), ('overheard', 7), ('plans', 204), ('nah', 18), ('chara
cter', 7023), ('ranted', 4), ('wanted', 1352), ('people', 9285), ('know', 6166), ('supplying', 10), ('etc', 1212), ('dunno', 24), ('hate
s', 104), ('lots', 799), ('things', 3688), ('turning', 344), ('car', 1225), ('robot', 217), ('whole', 3078), ('speed', 249), ('demon', 18
```

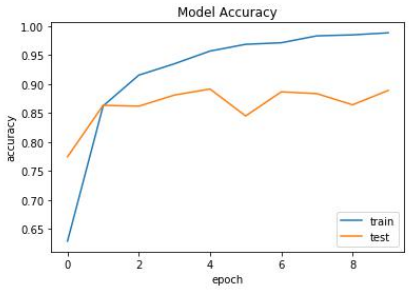
In [30]:
```
X_train = tokenizer.texts_to_sequences(train_text)
for i in range(3):
    print(X_train[i])
X_train = sequence.pad_sequences(X_train, maxlen=1416)
X_train, X_test, y_train, y_test = train_test_split(X_train, y, test_size = 0.05, random_state = 100)
```

```
[404, 70, 419, 8815, 506, 2456, 115, 54, 873, 516, 178, 18686, 178, 11242, 165, 78, 14, 662, 2457, 117, 92, 10, 499, 4074, 165, 22, 210, 58
1, 2333, 1194, 11242, 71, 4826, 71, 635, 2, 253, 70, 11, 302, 1663, 486, 1144, 3265, 8815, 411, 793, 3342, 17, 441, 600, 1500, 15, 4424, 185
1, 998, 146, 342, 1442, 743, 2424, 4, 8815, 418, 70, 637, 69, 237, 94, 541, 8815, 26055, 26056, 120, 1, 8815, 323, 8, 47, 20, 323, 167, 10,
207, 633, 635, 2, 116, 291, 382, 121, 15535, 3315, 1501, 574, 734, 10013, 923, 11578, 822, 1239, 1408, 360, 8815, 221, 15, 576, 8815, 22224,
2274, 13426, 734, 10013, 27, 28606, 340, 16, 41, 18687, 1500, 388, 11243, 165, 3962, 8815, 115, 627, 499, 79, 4, 8815, 1430, 380, 2163, 114,
1919, 2503, 574, 17, 60, 100, 4875, 5100, 260, 1268, 26057, 15, 574, 493, 744, 637, 631, 3, 394, 164, 446, 114, 615, 3266, 1160, 684, 48, 11
75, 224, 1, 16, 4, 8815, 3, 507, 62, 25, 16, 640, 133, 231, 95, 7426, 600, 3439, 8815, 37248, 1864, 1, 128, 342, 1442, 247, 3, 865, 16, 42,
1487, 997, 2333, 12, 549, 386, 717, 6920, 12, 41, 16, 158, 362, 4392, 3388, 41, 87, 225, 438, 207, 254, 117, 3, 18688, 18689, 316, 1356]
[232, 203, 3048, 3565, 7116, 317, 2, 405, 153, 19, 634, 10967, 11898, 8816, 1653, 1035, 3494, 232, 154, 314, 7116, 2701, 178, 2, 2349, 87, 1
111, 582, 217, 2219, 149, 73, 160, 626, 1035, 2882, 194, 642, 3316, 3464, 3869, 154, 405, 180, 155, 158, 79, 1, 19718, 2177, 1251, 68, 6828,
170, 281, 811, 1, 532, 10968, 4, 2003, 16, 36, 881, 1251, 376, 634, 314, 7116, 159, 2564, 1653, 1035, 3494, 232, 511, 143, 317, 20, 623, 462
8, 1251, 8974, 5471]
[2, 382, 2818, 4393, 3730, 598, 2184, 17788, 518, 4313, 9355, 1328, 837, 1012, 37249, 9355, 1435, 624, 46199, 11244, 4, 37250, 1328, 3102, 1
1245, 3, 714, 18690, 13004, 9550, 4592, 32129, 3542, 5590, 374, 2305, 86, 193, 19719, 8332, 2869, 1443, 924, 630, 13891, 6048, 4954, 279, 97
78, 8817, 3542, 1920, 8500, 2350, 10474, 1454, 2366, 2046, 2827, 2911, 899, 1569, 1217, 1208, 37251, 1791, 1416, 2366, 1362, 12248, 37252, 3
465, 1396, 37253, 409, 832, 22225, 46200, 22226, 146, 53, 201, 256, 6135, 16218, 55, 923, 1301, 1208, 13891, 6048, 4954, 459, 384, 16219, 40
9, 3, 714, 18690, 13004, 3900, 687, 9550, 1656, 413, 8818, 1308, 1, 1382, 6301, 627, 399, 455, 15536, 991, 4314, 5218, 244, 2425, 22226, 82
4, 1350, 198, 183, 9, 1403, 968, 8023, 571, 1750, 77, 237, 1301, 1208, 1999, 20, 1064, 18691, 191, 354, 2143, 17789, 233, 55, 1684, 12249, 5
048, 1409, 1472, 59, 95, 19720, 1578, 233, 15537, 13427, 481, 2552, 8666, 13005, 3227, 6218, 505, 344, 2305, 48, 266, 229, 672, 2030, 953, 1
109, 49, 30, 22226, 32129, 891, 9550, 439, 1078, 22227, 5591, 2064, 466, 4556, 182, 8975, 3300, 13, 44, 13428, 10724, 28607, 1975, 15538, 37
31, 3072, 16986, 7427, 1110, 304, 906, 1602, 262, 759, 392, 578, 2674, 2656, 187, 32, 645, 808, 30, 2674, 2251, 60, 493, 631, 551, 23948, 26
35, 1123, 2046, 534, 32, 281, 1875, 302, 43, 8501, 1671, 23949, 23950, 4356, 32130, 535, 700, 1175, 5310]
```

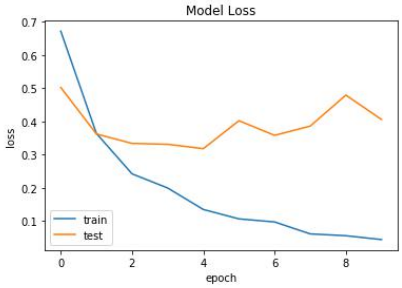
In [11]:
```
DROPOUT_RATE = 0.5
HIDDEN_UNITS = 64
model2=Sequential()
model2.add(layers.Embedding(max_features,100,mask_zero=True))
model2.add(layers.Bidirectional(layers.LSTM(64, return_sequences=False)))
model2.add(layers.Dropout(DROPOUT_RATE))
model2.add(layers.Dense(num_classes,activation='softmax'))
model2.compile(loss='categorical_crossentropy',optimizer=keras.optimizers.Adam(lr=1e-4),metrics=['accuracy'])
model2.summary()
history2=model2.fit(X_train, y_train, validation_data=(X_test, y_test),epochs=10, batch_size=BATCH_SIZE, verbose=1)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 100)         7406400
_________________________________________________________________
bidirectional (Bidirectional (None, 128)               84480
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 2)                 258
=================================================================
Total params: 7,491,138
Trainable params: 7,491,138
Non-trainable params: 0
_________________________________________________________________
Train on 23750 samples, validate on 1250 samples
Epoch 1/10
23750/23750 [==============================] - 12032s 507ms/sample - loss: 0.6711 - accuracy: 0.6288 - val_loss: 0.5019 - val_accuracy: 0.77
44
Epoch 2/10
23750/23750 [==============================] - 8511s 358ms/sample - loss: 0.3636 - accuracy: 0.8621 - val_loss: 0.3621 - val_accuracy: 0.863
2
Epoch 3/10
23750/23750 [==============================] - 8199s 345ms/sample - loss: 0.2422 - accuracy: 0.9151 - val_loss: 0.3337 - val_accuracy: 0.861
6
Epoch 4/10
23750/23750 [==============================] - 8233s 347ms/sample - loss: 0.1996 - accuracy: 0.9348 - val_loss: 0.3310 - val_accuracy: 0.880
8
Epoch 5/10
23750/23750 [==============================] - 8002s 337ms/sample - loss: 0.1350 - accuracy: 0.9567 - val_loss: 0.3179 - val_accuracy: 0.891
2
Epoch 6/10
23750/23750 [==============================] - 7638s 322ms/sample - loss: 0.1067 - accuracy: 0.9683 - val_loss: 0.4020 - val_accuracy: 0.844
8
Epoch 7/10
23750/23750 [==============================] - 7687s 324ms/sample - loss: 0.0975 - accuracy: 0.9710 - val_loss: 0.3580 - val_accuracy: 0.886
4
Epoch 8/10
23750/23750 [==============================] - 7828s 330ms/sample - loss: 0.0616 - accuracy: 0.9829 - val_loss: 0.3858 - val_accuracy: 0.883
2
Epoch 9/10
23750/23750 [==============================] - 7361s 310ms/sample - loss: 0.0561 - accuracy: 0.9846 - val_loss: 0.4790 - val_accuracy: 0.864
0
Epoch 10/10
23750/23750 [==============================] - 7267s 306ms/sample - loss: 0.0445 - accuracy: 0.9881 - val_loss: 0.4057 - val_accuracy: 0.888
8
```

In [31]:
```
fig = plt.figure()
plt.plot(history2.history['accuracy'])
plt.plot(history2.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower right')
plt.show()
```



In [32]:
```
fig = plt.figure()
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower left')
plt.show()
```



In [33]:
```
model2.save('72.h5')
```

In [34]:
```
x_test = ['This movie is excellent.','I love this movie.','It is boring.']
x_test1 = tokenizer.texts_to_sequences(x_test)
x_test2 = sequence.pad_sequences(x_test1, maxlen = 500)
labels = [int(round(x[1])) for x in model2.predict(x_test2) ]
label2word = {1:'Positive', 0:'Negative'}
for i in range(len(x_test)):
    print('{}    {}'.format(label2word[labels[i]], x_test[i]))
```

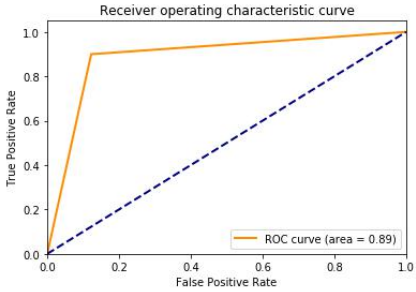
```
Positive   This movie is excellent.
Positive   I love this movie.
Negative   It is boring.
```

In [35]:
```
pred=model2.predict(X_test)
pred1=[]
for i in range(len(pred)):
    pred[i][0]=int(round(pred[i][0]))
    pred[i][1]=int(round(pred[i][1]))
    if(pred[i][0]<pred[i][1]):
        pred1.append(1)
    else:
        pred1.append(0)
y_test1=[]
for i in range(len(y_test)):
    if(y_test[i][0]<y_test[i][1]):
        y_test1.append(1)
    else:
        y_test1.append(0)
```

In [36]:
```
# accuracy of model
accuracy_score(y_test1, pred1)
```

Out[36]: 0.8888

In [37]:
```
fpr, tpr, thresholds = metrics.roc_curve(y_test1, pred1, pos_label=1)
#ROC curve
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',lw=lw, label='ROC curve (area = %0.2f)' % accuracy_score(y_test1, pred1))
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic curve')
plt.legend(loc="lower right")
plt.show()
```



In [38]:
```
# auc area
metrics.auc(fpr, tpr)
```

Out[38]: 0.8885899698064695

In [40]:
```
print(classification_report(y_test1, pred1))
```

	precision	recall	f1-score	support
0	0.89	0.88	0.89	613
1	0.88	0.90	0.89	637
accuracy			0.89	1250
macro avg	0.89	0.89	0.89	1250
weighted avg	0.89	0.89	0.89	1250

Appendix C:

The complete Python file code:

```
import numpy as np
from flask import Flask
from flask import request
from flask import jsonify
from sklearn.externals import joblib
from sklearn.feature_extraction.text import TfidfVectorizer as TFIDF
from flask import render_template
import pickle
import numpy as np
#import tensorflow as tf
#import pandas as pd

from tensorflow.keras.preprocessing import sequence,text
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout,Embedding,LSTM,Conv1D,GlobalMaxPooling1D
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.text import Tokenizer as word_tokenize
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,f1_score
from nltk.tokenize import word_tokenize
from nltk import FreqDist
from nltk.stem import SnowballStemmer,WordNetLemmatizer
from tensorflow.keras.models import load_model
from bs4 import BeautifulSoup
import nltk
import os
import gc
import re
from nltk.corpus import stopwords


app = Flask(__name__)

tf_model = joblib.load('./bayesmodel/tf_model.m')
clf_model = joblib.load('./bayesmodel/clf_model.m')
#import models
model = load_model("./lstmmodel/72.h5")
with open("./lstmmodel/tokenizer1.pk", "rb") as f:
      token_model = pickle.load(f)

stoplist = ['very','ourselves', 'am', 'doesn', 'through', 'me', 'against', 'up', 'just', 'her', 'ours','couldn', 'because', 'is', 'isn',
'it', 'only', 'in', 'such', 'too', 'mustn', 'under', 'their','if', 'to', 'my', 'himself', 'after', 'why', 'while', 'can', 'each',
'itself', 'his', 'all', 'once', 'herself', 'more', 'our', 'they', 'hasn', 'on', 'ma', 'them', 'its', 'where', 'did', 'll', 'you','didn',
'nor', 'as', 'now', 'before', 'those', 'yours', 'from', 'who', 'was', 'm', 'been', 'will','into', 'same', 'how', 'some', 'of', 'out',
'with', 's', 'being', 't', 'mightn', 'she', 'again', 'be', 'by', 'shan', 'have', 'yourselves', 'needn', 'and', 'are', 'o', 'these',
'further', 'most', 'yourself','having', 'aren', 'here', 'he', 'were', 'but', 'this', 'myself', 'own', 'we', 'so', 'i', 'does',
'both','when', 'between', 'd', 'had', 'the', 'y', 'has', 'down', 'off', 'than', 'haven', 'whom', 'wouldn','should', 've', 'over',
'themselves', 'few', 'then', 'hadn', 'what', 'until', 'won', 'no', 'about','any', 'that', 'for', 'shouldn', 'don', 'do', 'there', 'doing',
'an', 'or', 'ain', 'hers', 'wasn','weren', 'above', 'a', 'at', 'your', 'theirs', 'below', 'other', 'not', 're', 'him', 'during', 'which']

```

```python
@app.route('/')
def index():
      return render_template('index.html')   # render

@app.route('/bayes',methods=['GET','POST'])
def bayls():
      text = request.form['inputdata']
      if text:
            x_test = removeStopword(text)
            x_test = review_to_wordlist(text)
            x_test = [x_test]
            x_test = tf_model.transform(x_test)
            result = clf_model.predict(x_test)[0]
            restscore = clf_model.predict_proba(x_test)[0]
            return jsonify({"code":str(result),'score1':str(restscore[0]),"score12":str(restscore[1])})
      else:
            return 5
@app.route('/lstm',methods=['GET','POST'])
def lstm():
      text = request.form['inputdata']
      if text:
            x_test = removeStopword(text)
            x_test = review_to_words(x_test)
            x_test = token_model.texts_to_sequences([x_test])
            x_test = sequence.pad_sequences(x_test, maxlen = 1416)
            result = model.predict(np.array(x_test))
            restscore = result[0]
            index = np.argmax(result)
            return jsonify({"code":str(index),'score1':str(restscore[0]),"score12":str(restscore[1])})

#remove stop words
def removeStopword(x):
      return   str(' '.join([word for word in x.split() if word not in stoplist]))


#processing data by Naive Bayes model
def review_to_wordlist(review):
      #only save words
      review_text = re.sub("[^a-zA-Z]"," ", review)
      #to lower case
      words = review_text.lower()

      return(words)
#processing data by LSTM model
def review_to_words( raw_review ):
      #remove html labels
      review_text = BeautifulSoup(raw_review).get_text()
      #remove special characters
      letters_only = re.sub("[^a-zA-Z]", " ", review_text)
      #to lower case
      words = letters_only.lower().split()
      #remove stop words
      stops = set(stopwords.words("english"))
      meaningful_words = [w for w in words if not w in stops]
      wordnet_lemmatizer = WordNetLemmatizer()
      #lemmatize list of words and join them
      for word in meaningful_words:
            word = wordnet_lemmatizer.lemmatize(word,'v')
      return( " ".join( meaningful_words ))

if __name__ == '__main__':
      app.config['JSON_AS_ASCII'] = False
      app.run(host = '0.0.0.0',port = 5003,debug=True)
```

The complete index.html code:

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">
  <meta name="author" content="GaryWang">
  <meta name='renderer' content='webkit'>
  <meta http-equiv='X-UA-Compatible' content='IE=Edge,chrome=1' >
  <meta name="apple-mobile-web-app-capable" content="yes">
  <meta name="apple-mobile-web-app-status-bar-style" content="black">
  <meta name="format-detection" content="telephone=no">
  <meta name="format-detection" content="email=no">
 <title>Sentiment Analysis of Reviews</title>
<link rel="stylesheet" href="https://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">

</head>
<style type="text/css">
  .top10{
    margin-top: 30px
  }
  html,body{
    height: 100%;
  }
  .body{
    height: 100%;
  }
  .buttombut {
    position: fixed;
    bottom: 100px;
    width: 94%;
  }
  .tophead{

  }
  .ppp{
    background: #0e0e0e;
    height: 30px;
    line-height: 30px;
    color: white;
  }
  .restt {
    line-height: 30px;
    text-align: center;
  }
</style>
<body >
<div class="container top10 body">
    <div class="tophead">
        
    </div>
    <div align="center">
       <p class='text-muted ppp'>Please enter the content you want to analyse in the box below.</p >
    </div>
```

```
      <div  align="center">
      <textarea  class="form-control" style="height: 100px;">
      </textarea>
      </div >
     <div align="center"; class="top10">
      <select  class="form-control">
                 <option value="1">Naive Bayes Model</option>
                 <option value="2">LSTM Model</option>
      </select>
    </div>

    <div class='top10'>
      <p class='text-muted rest restt'></p >
      <p class='text-muted rest1 restt'></p >
      <p class='text-muted rest2 restt'></p >
    </div>

  <div class="buttombut">
    <button class="btn btn-success getsinglist" style="width: 65%;">
      Search
    </button>
  </div>
</div>
</body>
<script  src="https://code.jquery.com/jquery-3.4.1.min.js"  integrity="sha256-CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSF1Bw8HfCJo=" crossorigin="anonymous"></script>
<script type="text/javascript">
  $('.getsinglist').click(function(){
     var inputdata = $('textarea').val();
     inputdata=inputdata.trim();
     if(inputdata==""){
       alert("Please input the content!");
     }
     else{
       var model = $('select').val();
       var data = {inputdata:inputdata,model:model}
       var url = '/bayes'
       if(model == 1){
         url = '/bayes'
      }
       if(model == 2){
        url = '/lstm'
       }
     $.post(url,data,function(res){
       var text = 'Neutral'
        if(res.code==0){
          text = 'Negative'
        }
        if(res.code==1){
          text = 'Positive'
        }

       text = "The sentiment analysis result of the content you entered is: " + text
       s1=Math.round(res.score1*1000)/1000;
       s2=Math.round(res.score12*1000)/1000;
       $('.rest').text(text)
       $('.rest1').text("Negative: " + s1)
       $('.rest2').text("Positive: " + s2)


     })
   }

  })

</script>
</html>
```